修 士 論 文

マルチメディア通信に向けたラベルスイッチング方式の研究

指導教官　日比野 靖 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

張 偉

2000 年 2 月 15 日

要 旨

本研究では、可変長フレームを扱うスイッチング方式として、ラベルスイッチアーキテクチャを提案する。提案するスイッチは入、出力バッファと共有メモリ型のスイッチアーキテクチャの結合によって内部衝突が生じない理想型のスイッチができ、ソフトウェアスイッチング処理とハードウェア制御のクロスポイント出力バッファアレーの結合によってソフトウェア処理の割合を低減することができる。さらにマルチスレッド型プロセッサを用いて、ソフトウェア処理の高いスループットを期待することができるだけではなく、全ての入力回線からの長さが異なるフレームを一つの単純な循環バッファ方式の共有メモリに収容できる。ラベル技術を活用して、出力優先制御をハードウェアに任せ、QoS 制御が簡単かつ高速で行なわれる。また、入力バッファリングにおける閾値を設け、長さが異なるフレームの入力時間差を抑えることによって入力待ち時間を短縮できる。出力ッファリングにおける仲裁ロジックにより優先性と公平性を調整することもできる。

このようなスイッチアーキテクチャでは、スイッチング処理アルゴリズムの簡素化とソフトウェア処理の割合の低減を達成し、進んだプロセス技術を利用することによって、B-ISDN の基本インタフェースの要求を満たすことが可能である。

本稿では、このラベルスイッチアーキテクチャの構成とスイッチング処理アルゴリズム及びその実現可能性を示す。

目 次

第 1 章

序論

1.1 本研究の背景と目的

近年のデジタル通信技術の発達により音声、画像、高速データなどのトラヒックを一元的に扱うマルチメディア通信ネットワークが期待されている。それに伴い、マルチメディアトラヒックに向けた転送とスイッチング方式に対する要求も高まっている。データ通信の最も基本的な技術として、固定長方式と可変長方式の論争がくすぶり続けている。マルチメディアトラヒックは多重化の視点から見れば、固定長のデータパケットを扱う方式より可変長のデータパケットを扱う方が多様なトラヒックの特性に対して柔軟的な適合することができると認められるが、プロトコルの複雑さとストア・アンド・フォワードのスイッチング処理の複雑さによりソフトウェア処理に頼らなければならない。それによって、処理スピードの巨大な壁を乗り越えることが難しいと認められる。結局、固定長方式の代表として短いセルを扱う ATM 方式[1]が普及した。

ATM 技術は現在マルチメディア通信をサポートする高速のヘッダ駆動スイッチング（ラベルスイッチング）の唯一の技術として注目され、特にハードウェアで実現する高速スイッチが多数提案されている。しかし、現在の ATM 方式は様々な特性を持つマルチメディアトラヒックに適合した処理が困難である問題点も顕在化しつつある。例えば、伝送遅延に対する制限が厳しい音声を ATM で扱うために、フェローセルという VCI 共有セル多重化方式が提案されている [1]。一方、大容量データを扱うために、複数のセルで構成

[1] ATM 方式とは、Asynchronous Transfer Mode の頭文字をとった言葉で、非同期転送モードのことである。多重化とスイッチングの単位としてセルという 53 バイトの固定長データパケットになる。すなわち、全ての情報をセルに分割して転送される。ATM は統計多重をすることで、時分割多重より高い多重効率を得られ、個々の通信に割り当てる転送帯域を自由に設定できる。回線交換とパケット交換の長所を併せ持った技術という特徴から、B-ISDN の中核技術とされる。

される ATM ブロックを転送単位として効率的な多重化方式の研究も盛んでいる [2]。特にインターネットの IP パケットを ATM セルに分割して取り扱う IP over ATM という方式が一般化している。しかし、ATM では短いオクテットの固定長のセルしか扱うことができず、IP パケットを ATM で扱うのはオーバヘッドと損失率の増加が問題点となる [3]。

こういう状況に対して、トラヒックの特性に対してより柔軟に適合した可変長多重化とスイッチング方式の研究が増えている。特に、近年通信路の品質が著しい向上したことで、誤り制御などを低層から高層に委譲することが十分可能になり、それにより低層のプロトコルを簡素化して、高速処理をすることが十分可能になった。さらに、近年のチップ技術がすさまじく進歩し、高速かつ高スループットのプロセッサやメモリなどが次々と産み出された。RISC の登場と、その後のスーパスカラプロセッサや、スーパパイプラインプロセッサ等の高度のパイプラインプロセッサ（DEC Alpha 等）の登場によって、カスタム LSI との組合せで、今日の高速ルータやスイッチを支えている。

本研究の主題とは、可変長フレーム通信方式の検討ではなく、それを支える高速スイッチアーキテクチャの検討である。従って、可変長のフレームを扱う多重化とスイッチング方式を検討することに基づく、可変長フレームの高速スイッチング要求を満たすラベルスイッチアーキテクチャを提案し、その実現可能性を明らかにすることを狙っている。

1.2 本論文の構成

本論文は七章から構成される。

第一章では、本研究の背景と目的及び本論文の構成について述べる。

- 多種多様のトラヒックに対して、可変長方式は固定長方式より柔軟性があるが、高速スイッチング処理が困難である。
- 近年プロセッサ性能と通信路品質の著しく向上によって、可変長フレームを扱う高速スイッチの実現が可能になった。

第二章では、AFFTM フォーマットの提案について簡単に説明する。

- 可変長フレームを基本伝送単位として多重化とスイッチングを行なう。
- プロトコルの簡素化により高速処理を狙っている。
- AFFTM レイヤにおける QoS 制御を図る。

第三章では、ラベルスイッチアーキテクチャについて説明する。

- 共有メモリとクロスポイントを結合する方式で内部衝突が生じない理想的なスイッチを構築する。
- マルチスレッドプロセッサを用いることにより、ソフトウェア処理の高いスループットを期待する。
- クロスポイント出力バッファアレーにおける優先制御により AFFTM レイヤの QoS 制御を行なう。

第四章では、シミュレーションによる処理スループットの検証について述べる。

- I/O プロセッサの処理能力の評価。
- マルチスレッドプロセッサによるヘッダ処理能力の向上。

第五章では、シミュレーションによる入力バッファリングの検討について述べる。

- フレーム長によって、I/O プロセッサ直接制御と DMA 制御二つの入力方式を採る。
- 入力バッファリングにおいて、バッファサイズと待ち時間の検討。

第六章では、シミュレーションによる出力バッファリングの検討について述べる。

- 制限時間の役割の検討。
- 出力バッファリングにおいて、バッファサイズと待ち時間の検討。

第七章では、本論文のまとめを行なう。

- 提案した AFFTM 方式をまとめる。
- 提案したラベルスイッチアーキテクチャの特徴をまとめる。
- 機能検証の結論をまとめる。
- 本研究に関する問題点の提起、今後の課題を示す。

第 2 章

AFFTM 方式の提案

データ伝送の多重化方式は大まかに同期多重（いわゆる時間位置多重）と非同期多重（いわゆるラベル多重）に分けられる。同期多重では、異なるユーザのデータをそれぞれに一定の伝送時間を割り当てる。送るべきデータがなくても空のデータを送る必要がある。非同期多重ではヘッダで宛先を明らかにすることにより、必要な時に伝送を行なう。同期多重の交換方式は回線交換であり、非同期多重の交換方式はパケット交換である。パケット長で分類すると、非同期多重には固定長と可変長がある。

ネットワークの設計や運用において、データの伝送単位としてパケット長が固定か可変かについては長所、短所が存在する。固定長パケットの利点は、高速のスイッチング処理を実現しやすいことである。パケット長が全て同一ならば、パケット交換は同期的に行なわれ、ハードウェア設計が容易になる。また、各交換ノードでパケット長を計算、識別する必要がなく、処理形態が簡単なものとなる。一方、可変長パケットの主要な利点は、異なった通信サービスの性能要求に対し、適切なパケット長を選択できることである。例えば、音声のエンドーエンド遅延は、パケット長が短いほど好ましいものとなる。他方、LAN などのデータについては、一般に長いパケットの方が良好な性能を期待できる。また、パケットが長い場合、ヘッダによるオーバヘッドを軽減することもできる。つまり、異なった複数の通信サービスを単一ネットワークで伝送する際、各通信サービスの特性に動的に適合させるためには可変長パケットが有利となる。

本研究では、非同期可変長方式を選択した。

2.1 従来の多重化・スイッチング方式

スイッチング方式は表 2.1でまとめる。

	方式	特徴
回線交換	時間の位置情報に基づき交換 交換単位は 8bit 固定長 （ハードウェア処理が可能）	高速通信に対応可能 送信データがなくても回線を占有 非効率
パケット交換	パケットヘッダの情報に基づき交換 交換単位（パケット）は可変長 （ソフトウェア処理が必要）	高速化に限界があり、遅延時間が長い 送信データがない時伝送路を占有せず 高効率
ATM 交換	セルヘッダの宛先情報に基づき交換 交換単位（セル）が固定長 （ハードウェア処理が可能）	高速伝送、低遅延が可能 セルの送信は有意情報発生時のみ 高効率

表 2.1: 各スイッチング方式

パケット交換技術として、フレームリレー[1]を取り上げる。フレームリレーの大きな利点は回線交換より多重化が柔軟であることで、つまり自由な通信帯域幅の設定、通信中の帯域幅変更、１本アクセス回線を用いて複数コールの多重化が可能である。従来のパケット交換より、プロトコルの簡素化によって低遅延時間特性を得られ、2Mbit/s 程度までの高速通信が可能である。特にレイヤ３及びレイヤ２上位サブレイヤのプロトコルを規定していないため、エンドシステム側では多様なプロトコルを持つLAN の接続、音声、映像転送等マルチメディア通信等のユーザ独立のプロトコルが使用可能となる。

ATM 方式の主な特徴も幾つかが取り上げられる、まず狭帯域から広帯域通信サービスの提供、つまり高速から低速まであらゆる速度の情報転送に適している。次に単一ネットワークで全ての通信サービスの実現、これはマルチメディアトラヒックに適している。さらに個々のユーザへの帯域割り当てが柔軟に設定でき、通信帯域の動的な割り当ても可能、これはバースト型トラヒックに対しても有効である。

2.2 AFFTM 方式の提案

本研究では AFFTM（Asynchronous Fast Frame Transfer Mode）という方式を提案する。AFFTM 方式の機能は殆んど ATM 方式から継承したもので、但し、基本伝送単位は

[1] フレームリレーは一種のパケット通信である。従来行なわれてきた X.25 プロトコル処理のうち、レイヤ３及びレイヤ２上位サブレイヤである論理リンク制御機能（再送制御など）を省略し、フレームの多重・分離及び転送誤り等のレイヤ２コア機能のサブセットでありデータリンクコアのみを行なう方式である。

フレームリレーはコネクション型のサービスを提供する。自由な通信帯域幅の設定と変動に対応することによって、マルチメディアの対応が可能である。1992 年ITU-T 勧告完了で、国際標準化された。

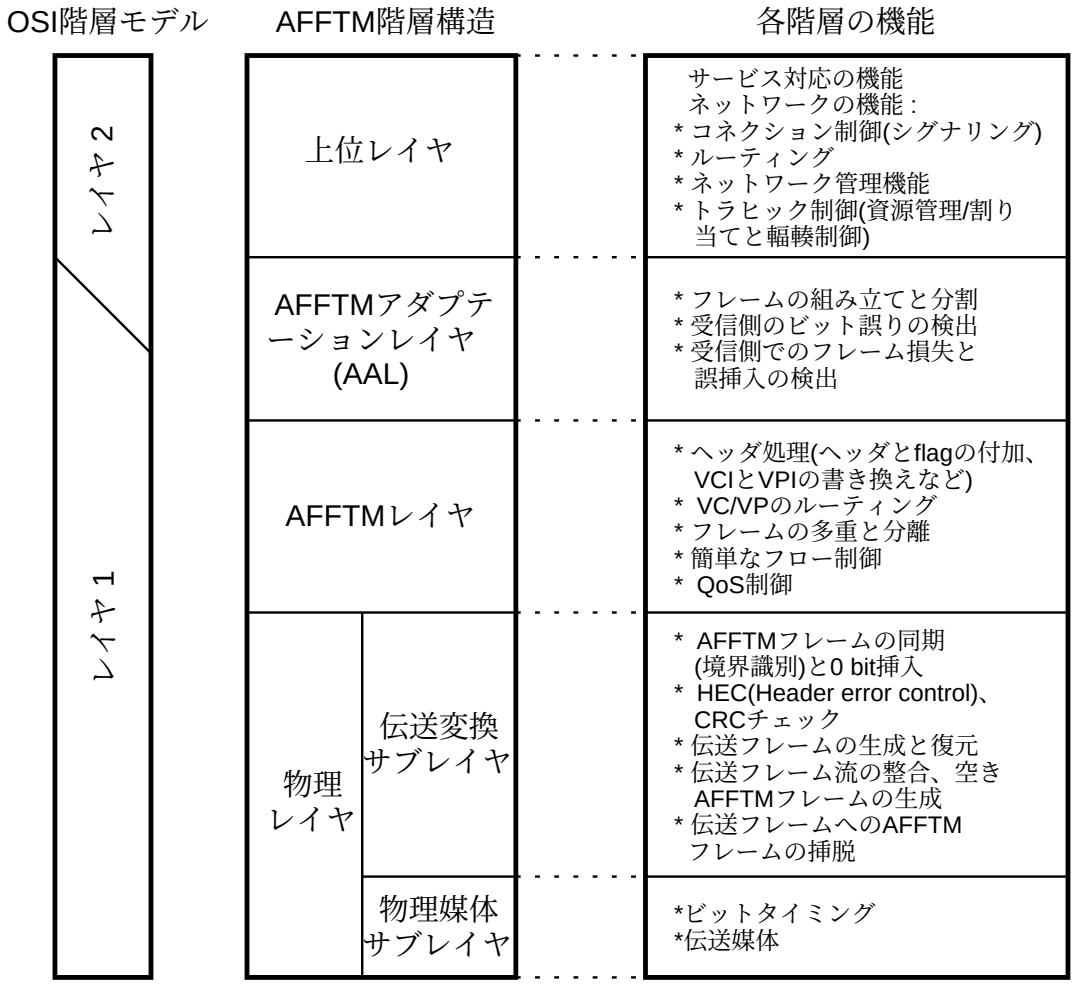


図 2.1: AFFTM 方式各レイヤの機能

可変長フレームである。AFFTM の各レイヤの機能を図 2.1で示す。

提案する AFFTM のフレームフォーマットを図 2.2で示す。

AFFTM フレームは 2 バイトのフラグと 7 バイトのヘッダ及び 48 バイトから 5111 バイトまでの可変長ユーザデータから構成し、ATM セルとの違いは遅延優先度とパイロード長を表すビットの導入である。フレームリレーとの主な違いはヘッダ部分だけ誤りチェックを行なう点である。これはプロトコルの簡素化により高速処理を実現することを狙っているからである。

最小パイロード長が ATM と同じであるため、音声トラヒックに対して、多重化遅延は同じであるが、複数ユーザが共有フレームで多重化する場合、かなり柔軟性がある。最

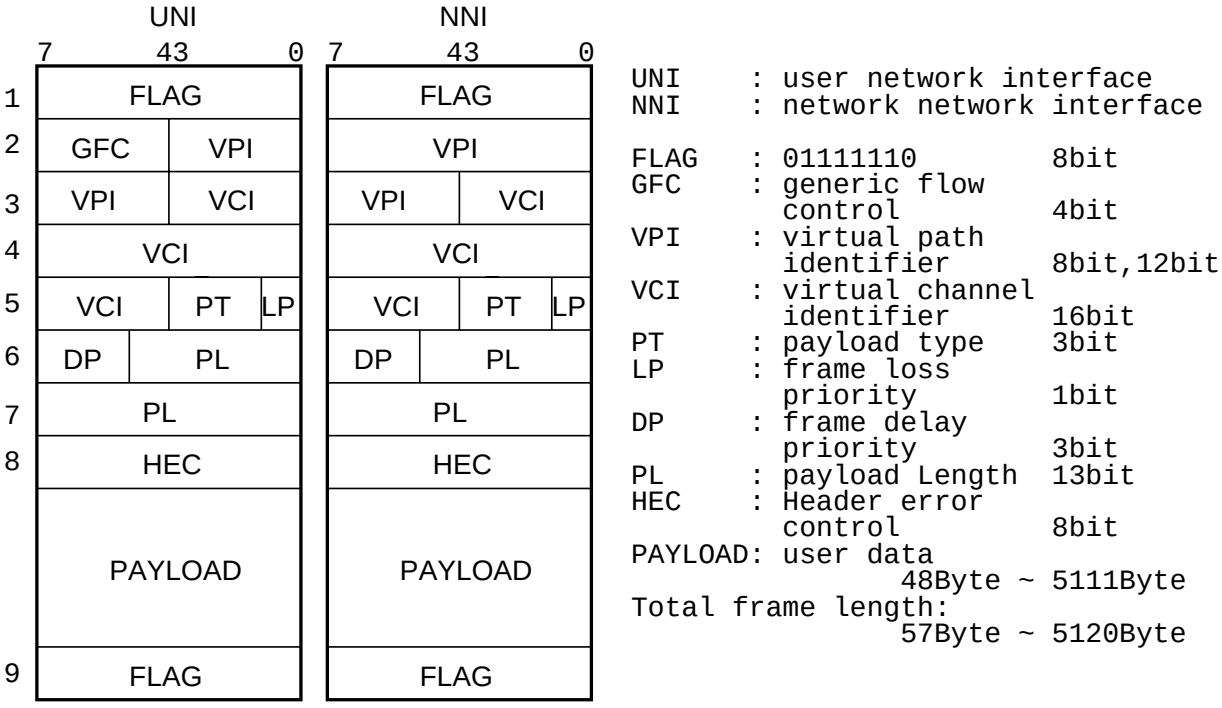


図 2.2: AFFTM フレームフォーマット

大フレーム長は 5KByte と決めるのは、データトラヒックに対して、オーバヘッドが小さい、特に可変長であるから、LAN のパケットをそのままフレームに載せることが可能である。

可変長フレームの不利な点は、フレームの先頭と末尾の検出処理が必要とされるため、フレームを送受する部分のハードウエアが複雑になることである。またフレームの内部にフラグパターンと同じパターンが現れることを避けるために、0bit の挿入が必要になる。これによってオーバヘッドが生じる。その結果、運ぶべきユーザ情報の内容によって、0bit の挿入が頻繁に行なわれ、見かけ上の情報量が増加して、物理的な情報伝送容量を越えてしまうため、情報を伝達できないことが生じ得る。

本研究の主題は可変長フレームを扱うラベルスイッチアーキテクチャの検討することであるため、AFFTM 方式の検討を AFFTM レイヤに限ることにした。

第 3 章

ラベルスイッチアーキテクチャ

AFFTM スイッチの基本的な仕事は、任意の入力回線の任意のバーチャルチャネルのフレームを要求によって任意の出力回線の任意のバーチャルチャネルにスイッチングすることである。このスイッチングは二つの意味を持つ、一つは空間的な交換、つまりある回線から別の回線に送るということ。もう一つは時間的な交換、つまりあるタイムスロットから別のタイムスロットに移すということである。AFFTM は非同期方式であり、バーチャルチャネルとタイムスロットの間に固定的な関係がないので、バーチャルチャネルの身分はフレームヘッダにより識別され、そして、時間的な交換はヘッダの情報によって行なう。

AFFTM は非同期方式であるので、バーチャルチャネルにフレームが現れるのはランダムである。これによって、スイッチにおける衝突の可能性がある。つまり、同じ時刻で二つ以上の入力回線上のフレームが同一出力回線へ行きたいということである。そして、その衝突を回避するため、スイッチの中にバッファを設け、フレームの待ち列を作る必要がある。

一言で言うと、AFFTM スイッチはルーティング、ヘッダ処理、バッファリング三つの機能を持ったなければならない。

3.1　各スイッチ構成法の簡単説明

非同期方式のスイッチの構成法と言えば、まず ATM スイッチが取り上げられる。ATM スイッチは入力バッファ方式、出力バッファ方式、共通バッファ方式とクロスポイントバッファ方式で大まかに分類される [9]。各構成を図 3.1出示す。

入力バッファ方式は、入力されるセルをいったんバッファに蓄え、出力回線空いている時順次セルを送り出す。この方式はバッファメモリのスピードへの要求を満たすのはやさ

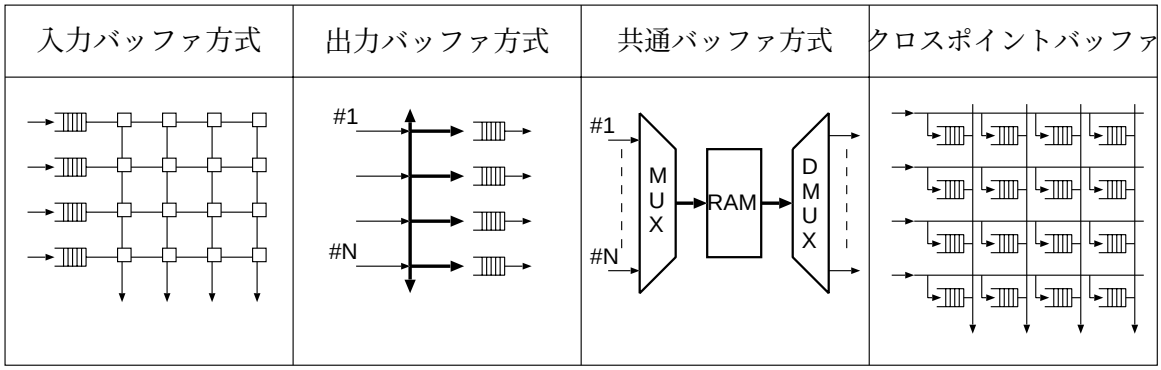


図 3.1: スイッチの構成法

しいが、同時に複数の入力バッファの先頭に同一宛先のセル到着する場合、先頭セルにより後ろのセル送出が妨げられる（Head of line blocking）という大きな欠点がある。

出力バッファ方式は出力ポート対応にバッファを配置する。複数の入力ポートから同一の出力ポートへ向かうセルが到着しても衝突が起きないよう、入力セルは高速バスに多重化する。そして回線速度の N 倍の速度で高速にバッファへ書き込む。この方式の欠点はバッファメモリのスピードの要求が厳しいことである。

共通バッファ方式はバッファを N 本のポートで共通利用するものであり、入、出力バッファ方式に比べバッファ数を削減でき、バッファサイズも小さい。共通バッファ方式の欠点は高い動作速度が要求される。その上制御ロジックが複雑で、実現が難しい。

クロスポイントバッファ方式は、入力ポートと出力ポートの交点にバッファを配置する。バッファメモリへの書き込みと読み出しは回線速度を同一の速度でよいため、高速な回線を扱うことが容易な反面、N^2個のバッファが必要のため、大規模化の制約となる。

3.2 ラベルスイッチアーキテクチャの構想

本研究では、非同期と可変長方式のフレームを扱うため、スイッチング処理はソフトウェア処理が必要となる。そして共有メモリを配置することにする。しかし、ATM 共通バッファ方式と違うのは、そこではフレームのバッファリングためではなく、フレーム処理のためだけに用いられる場所だけである。また、到着したフレームの長さが異なるため、入力バッファリングをしなければならない。この入力バッファリングは待ち列のためではなく、処理の基本単位として、フレームの塊を作る目的である。さらに、ソフトウェア処理の割合を低減することを狙って、出力優先制御をハードウェアに任せるため、出力

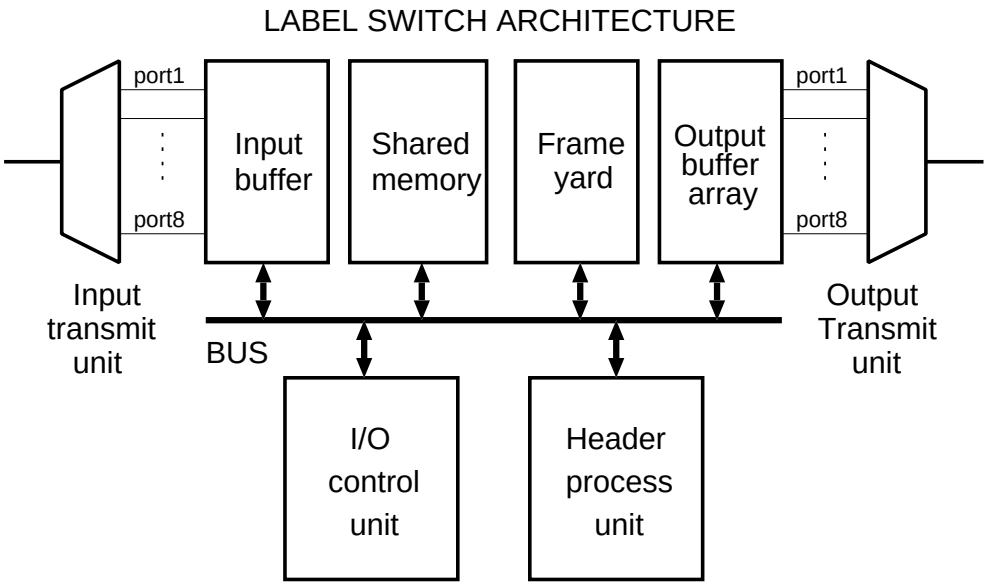


図 3.2: ラベルスイッチの構成

バッファも設けることにする。そこで正味のフレームのバッファリングをする。こういう複雑な仕組みになるので、処理のスピードに対して、かなり実現が難しい。しかし、処理方法のいろいろな工夫によって、ある程度の処理スピードとスループットを期待することができる。

本提案では出力バッファと共有メモリ型のスイッチアーキテクチャの結合によって内部衝突が生じない理想型のスイッチが考え、ソフトウェアスイッチング処理とハードウェア制御のクロスポイント出力バッファアレーの結合によってソフトウェア処理の割合を低減することを期待する。さらにマルチスレッド型プロセッサを用いて、ソフトウェア処理の高いスループットを狙っている。また、出力優先制御をハードウェアに任せ、QoS 制御を簡単かつ高速で行い、出力仲裁ロジックにより出力の優先性と公平性を調整することを目指している。

3.3 ラベルスイッチの構成とスイッチ処理アルゴリズム

3.3.1 ラベルスイッチの構成

図 3.2示しているように、提案するラベルスイッチは八つの部分で構成する。図 3.2の上の方はフレームが通る部分であり、両側のトランスミットユニットは主に物理レイヤの

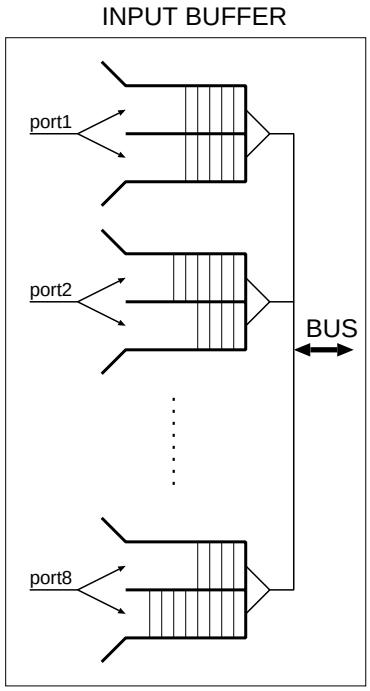


図 3.3: 入力バッファ

役割を担っている。物理レイヤの機能を果たしている LSI チップが多数存在しており、この部分はハードウェアで実現することができる。本研究は主に AFFTM レイヤのスイッチング機能を検討することに主眼をおいているので、トランスミットユニットの検討はしなかった。そして、残った六つの部分を説明していく。

1. 入力バッファ

   図 3.3で示しているように、一つの入力回線に対して、二つのバッファを設け、一つのバッファの中のフレームを共有メモリに運ぶ際に、後続のフレームは他の一つバッファに入れられる。二つのバッファが交替に使われ、到着したフレームをバッファリングする。そのバッファの容量を第五章で検討する。

2. 共有メモリ

   図 3.4で示しているように、共有メモリの中にアドレスポインタキューと幾つかの旧ヘッダキューと新ヘッダキューを設ける。これは可変長フレームのヘッダ処理とスイッチラベルを張り付けるなどの仕事を行なう場所である。

   入力処理の時、入力バッファから運ばれたフレームがまず分割され、ヘッダ部分を旧ヘッダキューに入れ、ボディー部分をフレームヤードに入れる。そして、そのフ

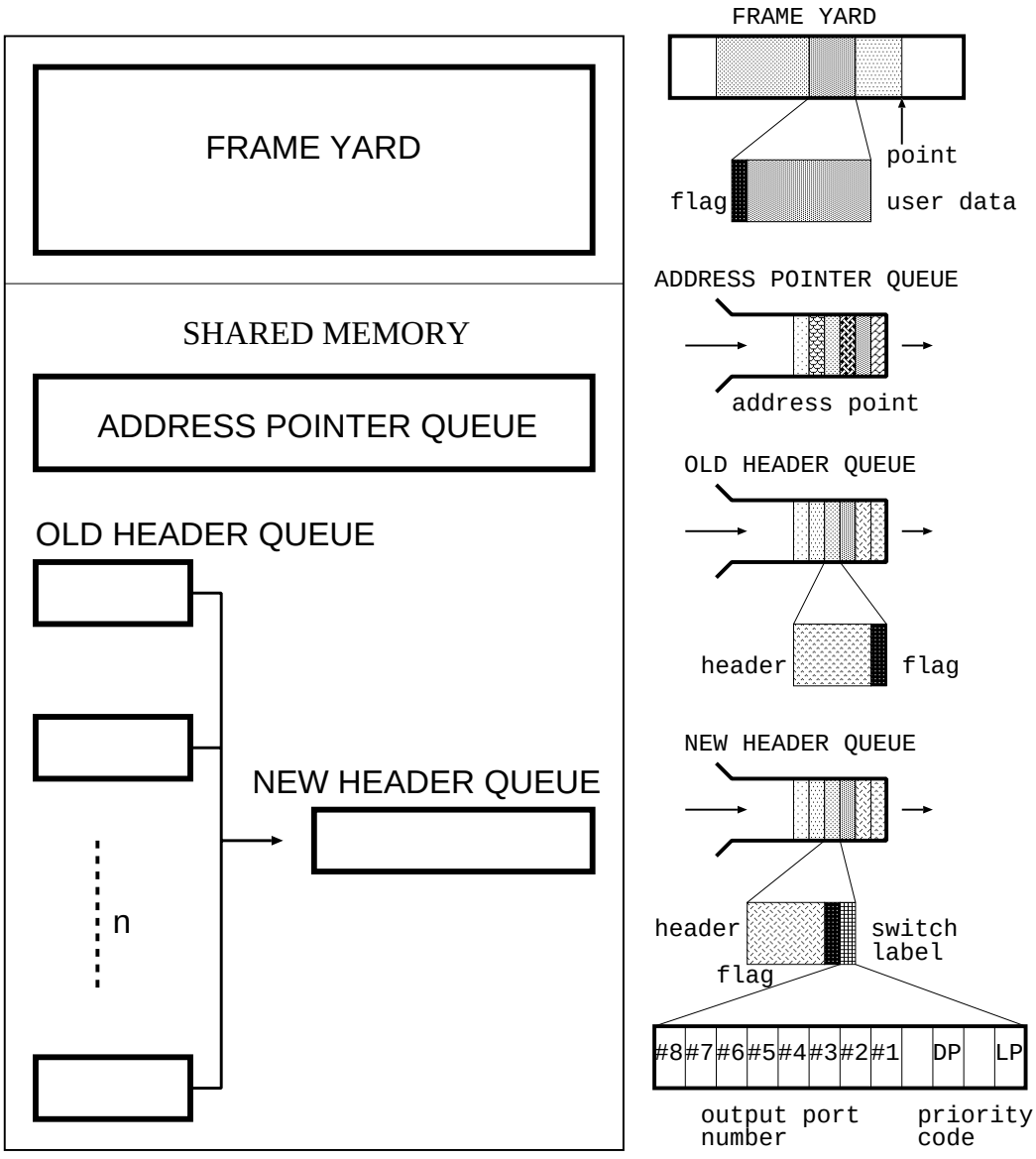


図 3.4: 共有メモリとフレームヤード

レームのボディー部分がフレームヤードの中のある場所を指すポインタをアドレスポインタキューに入れる。

ヘッダ処理の際、フレームのヘッダ部分を旧ヘッダキューから取り出し、ルーティング情報の書き換え等処理を行なう。処理終ったのヘッダの前にスイッチラベルをつけて、新ヘッダキューに入れる。

出力処理の時、新ヘッダキューから処理終ったヘッダを取り出し、そしてアドレスポインタキューからポインタを取り出し、そのポインタによってそのフレームのボディー部分の場所が解る。次にボディー部分をフレームヤードから取り出して、ヘッダ部分と合わせて、出力バッファアレーに投げる。

処理終ったヘッダの前に付けられたスイッチラベルは本提案の重要なポイントの一つである。このラベルには出力回線番号と優先度情報を含んでいるので、出力バッファアレーに行く時、大事な通行証となっている。図 3.4 の右側で、フレームヤード、アドレスポインタキューとヘッダキューの中身を示している。

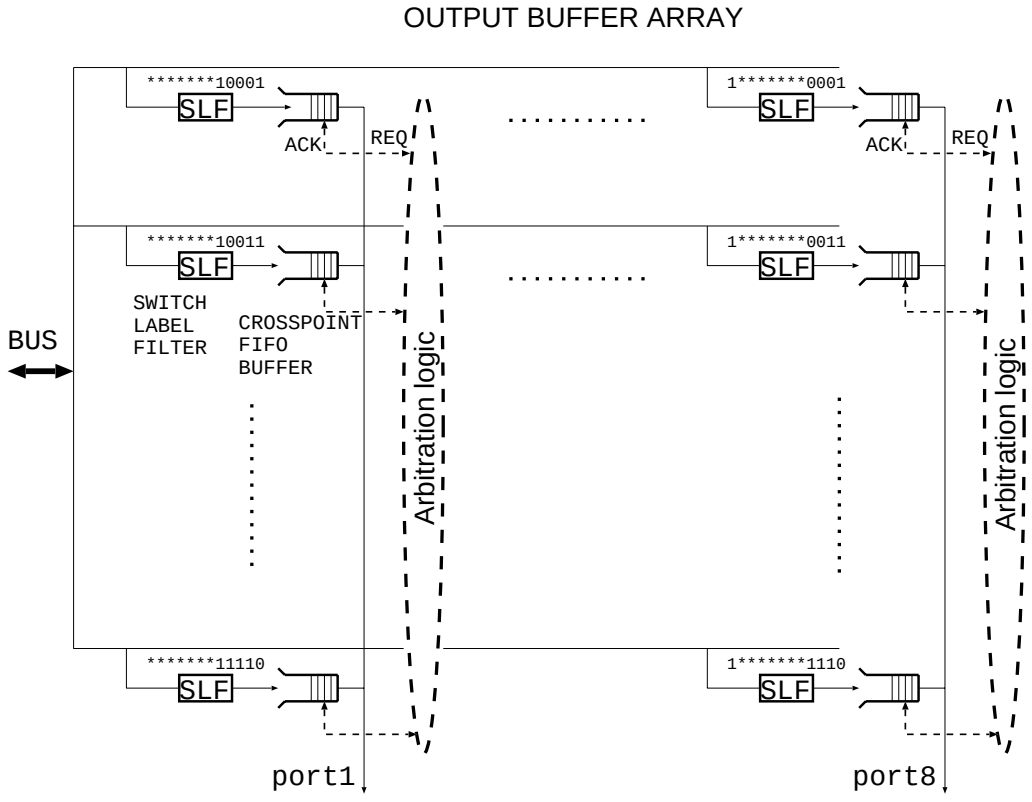


図 3.5: 出力バッファアレー

3. フレームヤード
   先ほど述べたように、フレームヤードはフレームのボディー部分を一時的な保存する場所である。

4. 出力バッファアレー
   出力バッファアレーはクロスポイントのストラクチャーである。しかし、列の方は出力回線と対応して、行の方はサービスの優先度のクラスと対応している。ATM クロスポイントバッファ方式との区別はバッファ数が N^2ではなく、出力回線 N×サービスクラス数である。したがって、バッファ数は大きな数ではない。

   出力バッファアレーに到着したフレームは頭にスイッチラベルを付いているから、その出力回線番号と優先度情報によって自律的に適当なフィルタを貫通して、適当な出力バッファに入り、そして出力リクエストを出して、出力されるのを待つ。その際、スイッチラベルは使命が完了したので、捨てられる。次に出力実行機構は仲裁ロジックによって、フレームの優先度と待ち時間に基づく出力制御を行なう。これによって、出力の優先性と公平性の両立を求める。仲裁ロジックは本提案の重要なポイントのもう一つである。これは後で詳しく説明する。

   また、このような仕組みは、ブロードキャストサービスに対してかなりメリットがあると考えられる。本研究ではそれを詳しく検討しなかったが、図 3.4で示したスイッチラベルのビット配置ではこれを少し考慮した。つまり出力回線番号のビットを全部１にすると、全てのフィルタを貫通すことができ、全ての回線に出力ができる、これによってフレームのコピー効果が得られる。

5. 入出力制御部
   入出力制御部は汎用プロセッサが担当する。入出力制御部の仕事は入力バッファからフレームを共有メモリに運び、処理終ったフレームを共有メモリから出力バッファアレーに運ぶことである。つまり、先ほど述べた入力処理と出力処理のことである。図 3.6では、入出力制御部の構成を簡単に示す。

6. ヘッダ処理部
   ヘッダ処理部の仕事は旧ヘッダキューからヘッダを取り出し、処理し、スイッチラベルを付け、新ヘッダキューに入れる、つまり先ほど述べたヘッダ処理のことである。ヘッダ処理部の操作範囲は共有メモリに限られる。ヘッダ処理のように複数の対象に対して、同じ処理を行なう場合は、複数のハードウェアコンテクストを持つ

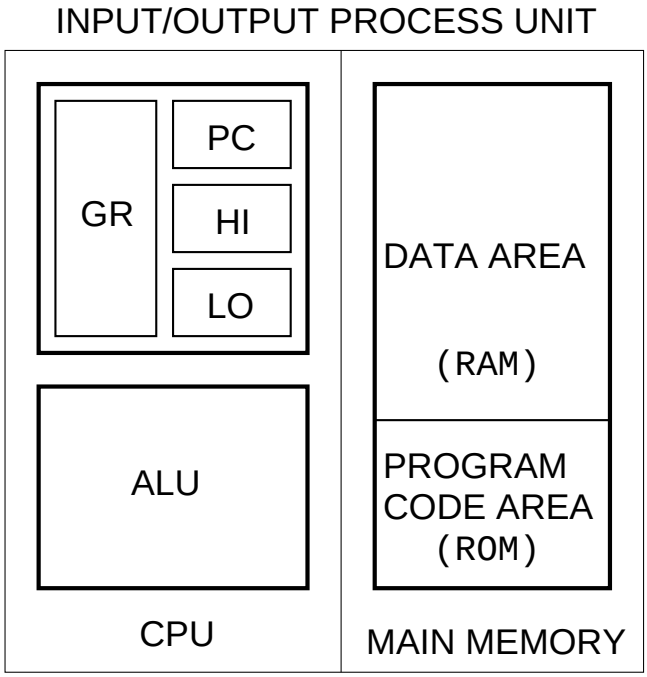


図 3.6: 入出力制御部

マルチスレッド型プロセッサ[1]は良く適合するので、ヘッダ処理部を旧ヘッダキューの数と同数のスレッドを持つマルチスレッド型プロセッサが担当することにする。また、スレッド数はプロセッサのパイプラインステージ数と同じであるので、パイプライン実行のハザードを解消することができ、プロセッサの処理能力を最大限度に引き出して、高いスループットのスイッチングパーフォマンスを期待できる。

図 3.7で複数ハードウェアコンテクストを持つマルチスレッド型プロセッサの様子を示している。マルチスレッド型プロセッサの使用は本提案の重要なポイントのもう一つである。これは後で詳しく説明する。

[1] 複数の命令を少しずつずらして同時並行的に実行する方式として、パイプライン実行はプロセッサの性能向上の重要手段として認められる。通常のパイプラインでは単一スレッドをパイプライン化しており、パイプラインハザードの問題が避けられず、効率的なパイプライン処理が妨げられる。マルチスレッド型プロセッサでは、ステージ数と同数の独立なスレッドの命令をパイプラインに投入し、全てのパイプラインステージを異なるスレッドの命令で埋めることで、パイプラインハザードをなくし、パイプラインをストールさせることがない。プロセッサ性能を最大に発揮することができる。

ハードウェアコンテクストを複数持つマルチスレッド型プロセッサは、コンテクストスイッチのオーバーヘッドも無しに、クロック毎にスレッドを切替えることが可能になる。

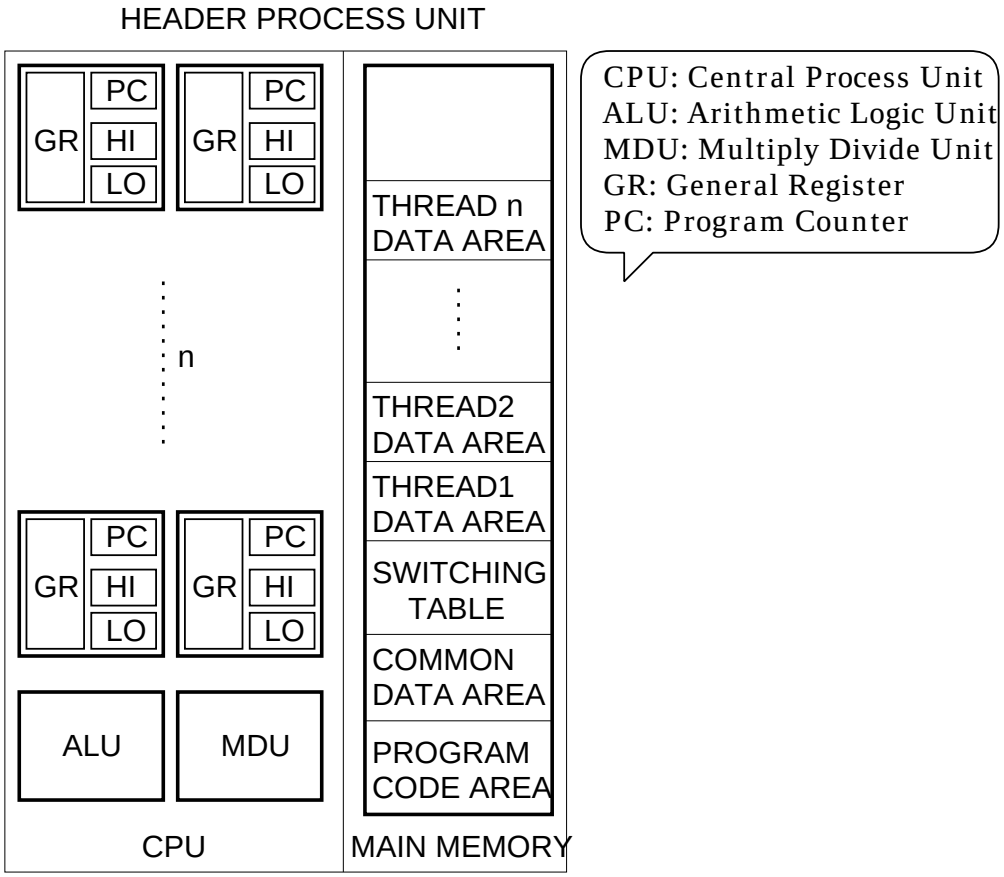

HEADER PROCESS UNIT
GR
PC
HI
LO
GR
PC
HI
LO
n
GR
PC
HI
LO
GR
PC
HI
LO
ALU
MDU
CPU
THREAD n
DATA AREA
THREAD2
DATA AREA
THREAD1
DATA AREA
SWITCHING
TABLE
COMMON
DATA AREA
PROGRAM
CODE AREA
MAIN MEMORY
CPU: Central Process Unit
ALU: Arithmetic Logic Unit
MDU: Multiply Divide Unit
GR: General Register
PC: Program Counter

図 3.7: ヘッダ処理部

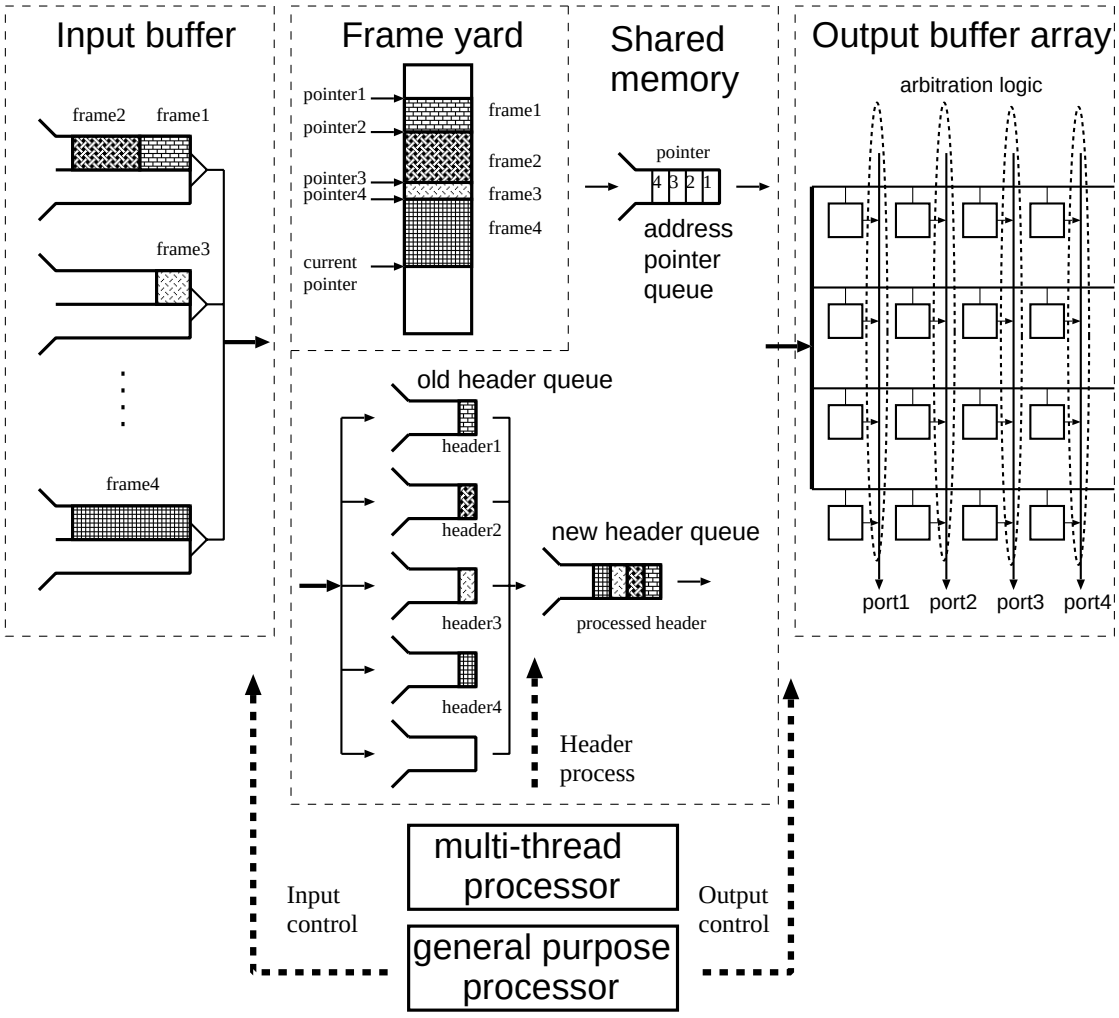


図 3.8: 基本処理アルゴリズム

3.3.2 スイッチ処理アルゴリズム

1. 基本処理アルゴリズム
   続いて図 3.8を用いて、基本処理アルゴリズムを説明する。

   まず、入力バッファリングにおいて、ある閾値を設けることにする。到着した長さが異なるフレームを入力バッファに貯める。これらのフレームの長さが合わせてその閾値を越えると、そのバッファの到着フラグを立てる。そして、I/O プロセッサはこのフレームの塊を共有メモリに運び、入力処理を行なう。フレームを分割した後、ヘッダ部分を順番に幾つかの旧ヘッダキューに入れる。

一方、ヘッダ処理プロセッサは最初から、全てのスレッドが立ち上がって、一つのスレッドは一つの旧ヘッダキューの面倒を見る。全てのスレッドが全く同じ操作なので、処理時間も同じであるから、先に旧ヘッダキューに入れられたフレームヘッダの処理を必ず先に済ませて、新ヘッダキューに入れられる。そうすると、フレームの処理順番を守ることができ、全てのフレームのボディー部分を一つのフレームヤードに収容することができる。それによって、スレッド間の仕事の負荷のバランスもとれ、処理操作もはるかに簡単になる。マルチスレッドプロセッサを用いて、プロセッサの処理能力の向上だけではなく、処理の手間も軽くなる。それは重要なポイントの一つになる。

続いて、I/O プロセッサは処理終ったフレームを出力バッファアレーに出す。

先ほど述べたように、入力バッファリングにおける閾値はI/O プロセッサの処理単位となるので、その閾値と入力回線速度によって処理時間が限られる。それで、I/O プロセッサはその限られた時間内で全ての回線の片側のバッファの処理を済ませ、相応する量の出力処理も済ませなければいけない。それに対して、ヘッダ処理プロセッサは相応するヘッダ処理能力も持ったなれればならない。

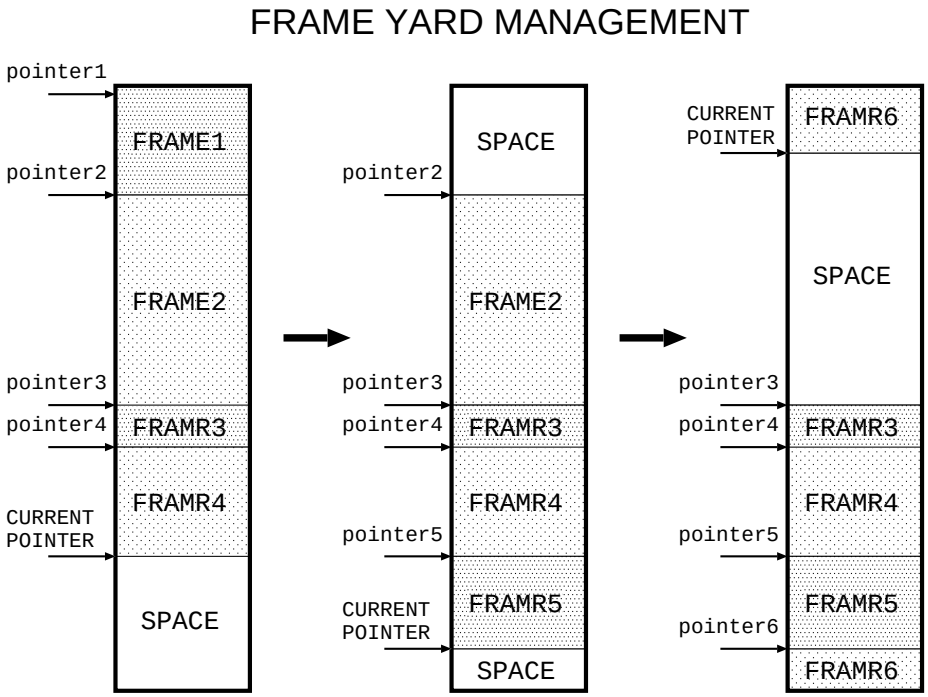


図 3.9: フレームヤードの管理方式

2. フレームヤードの管理
   フレームボディー部分の長さがそれぞれ異なるので、フレームヤードの管理が問題になる。本提案では図3.9のような循環バッファ方式を採る。

   カレントポインタは常に最後に入れたフレームボディーの末尾に指す。そして続いて入れたフレームボディーがそこから入れる。フレームヤードの底に当たると、裏回してトップからいれる。フレームボディーの書き込みと取り出しは同時に行なうので、そのフレームヤードのサイズが処理単位時間内の全ての入力回線から入ったフレームの長さの合計値より多ければ、収容するこのができる。つまり、図3.9で示すように、ある程度のスペースを守れば、順調に貯めることができる。この方式により異なるサイズのデータの一時的な保存は非常に簡単である。

3. 出力仲裁ロジック
   出力バッファに到着したフレームを出力回線に出す順番を決めるのは仲裁ロジックの仕事である。全ての回線の出力機構は全く同じなので、ここは一つの回線だけ図3.10で示している。

   フレームがある出力バッファに到着した時点で、そのフレームに対する出力リクエスト情報をそのバッファのホームラッチ内で生成する。その情報はバッファ番号と時間カウンタだけである。バッファ内のフレームの並ぶ順番と同じくホームラッチ内で並ぶ。その時間カウンタはフレームの待ち時間を表す。

   選択操作はステップというペースで行なう。仲裁は分散方式を採る。つまり各ステップにおいて、各レイヤの左ラッチと右ラッチの間に一つの出力リクエストが選ばれ、下のレイヤの右ラッチに進む。これは桁あげ操作と似ている。その選ぶルールとはバッファの優先度とフレームの待ち時間によって選ぶということである。あるレイヤの右ラッチの出力リクエストを桁あげると、上のレイヤから桁あげをもらう。左ラッチの出力リクエストを桁あげると、同レイヤのホームラッチから桁あげをもらう。このような選択はステップ・バイ・ステップで行なわれ、各ステップは必ず一つ決められた出力リクエストが実行機構に渡される。そして実行機構はその出力リクエスト情報に従って対応するバッファに出力許可を出す。当該バッファが自分の中の一番先頭のフレームを出力回線にリリースする。出力終った後、当該バッファが実行機構にレポートを送る。すると、実行機構は全てのラッチにステップ信号を出して、次の出力リクエストを獲得する。ちなみに、各ラッチは実行機構からのクロック信号を受け、出力リクエスト情報中の時間カウンタを増やす。

   選ぶルールとは以下の通りである：まずある閾値として制限時間を設けることにす

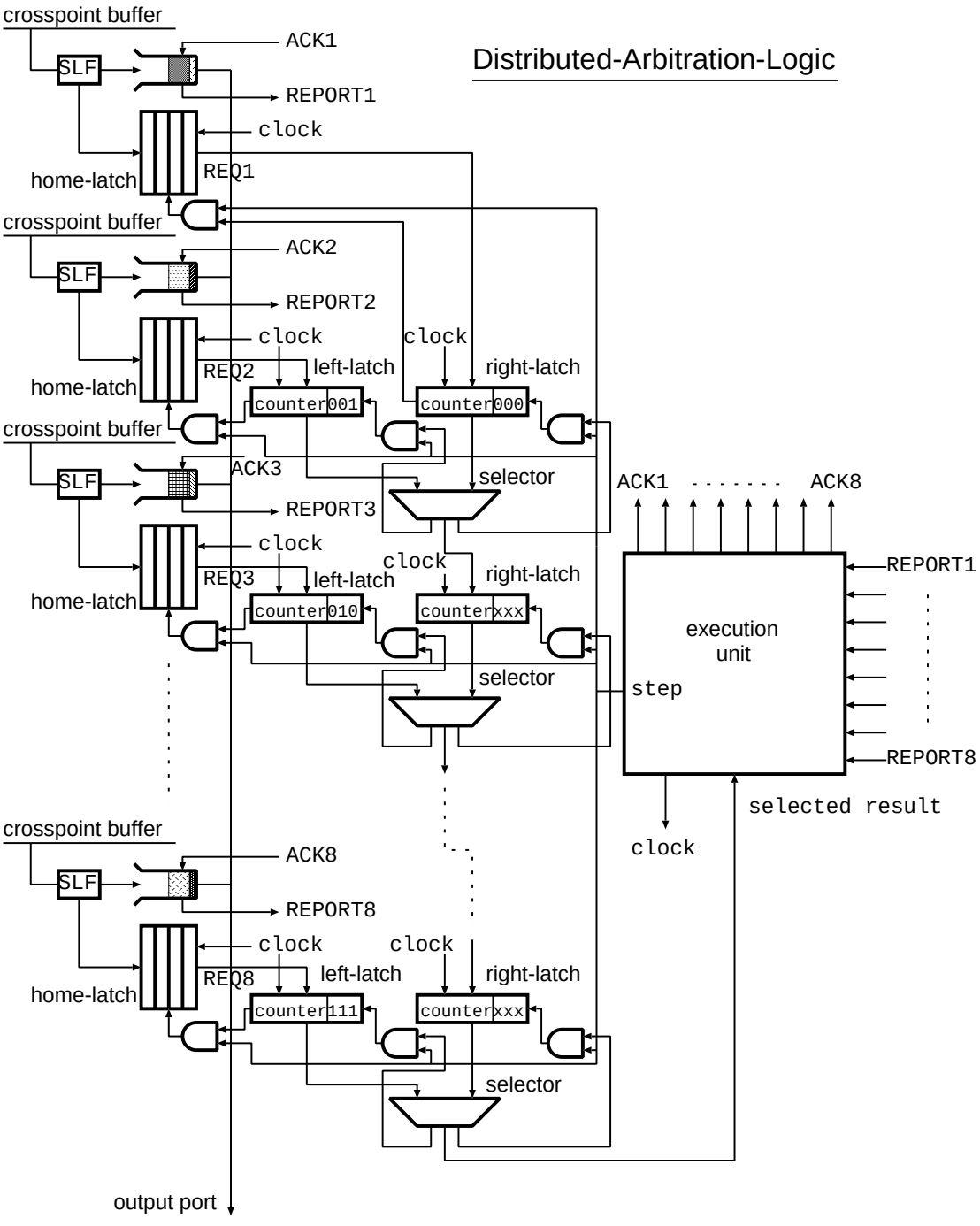
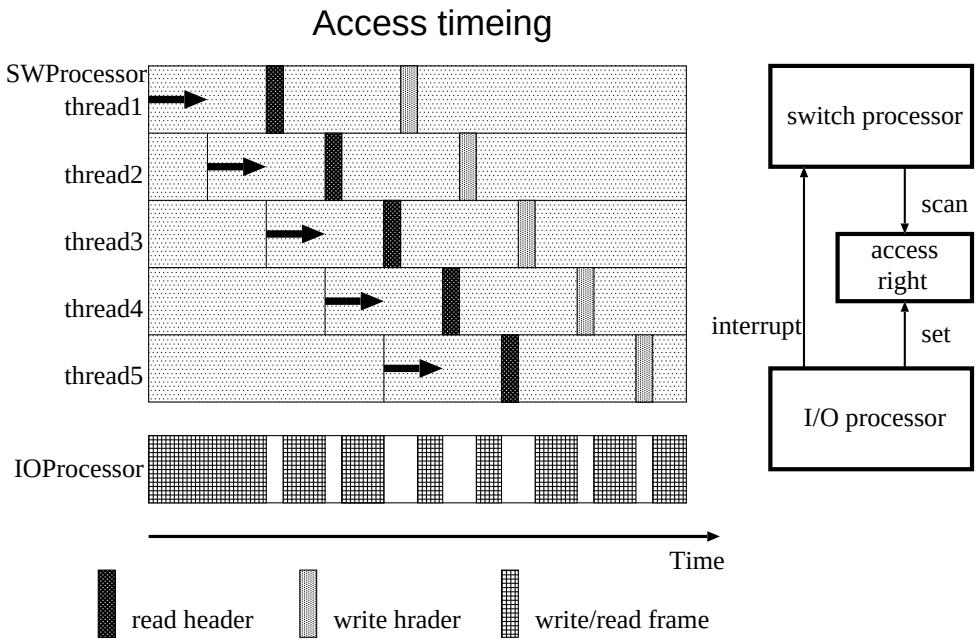


図 3.10: 出力仲裁ロジック

る。ある出力リクエストの時間カウンタ値がこの制限時間値を越えると、そのフレームが既に長い時間で待っていたと認められ、そのフレームを優先して選ぶことになる。全てのレイヤの二つのラッチの状態は次の幾つ場合がある。

- 左ラッチと右ラッチのカウンタが共に制限時間を越える場合、左ラッチを選ぶ。
- 一方のラッチのカウンタが制限時間を越える、他の一方が越えない場合、越えるの方を選ぶ。
- 両方とも越えないと、左ラッチを選ぶ。
- 二つのラッチの一方しか出力リクエストがない場合、勿論出力リクエストがあるラッチを選ぶ。

図 3.10を見ると解るように、下のレイヤのバッファの優先度が高い。こういう風に出力順番を決めることにより、優先性と公平性のバランスを調整でき、本提案の重要なポイントの一つになる。さらにこのような仲裁ロジックは従来の環状仲裁ロジックより、入力線（サービスクラス数）と出力スピードとの制限がなくなって、高速出力が可能を考えられる。



4. 共有メモリアクセス制御
   入出力制御部とヘッダ処理部両方は共有メモリをアクセスするので、衝突を避けるために、アクセス権の管理が必要になる。本提案では、次の方式を採る。普段はヘッダ処理部がアクセス権を保持している。出力制御部は共有メモリをアクセスしたい時、ヘッダ処理部に割り込みを行う。図3.11を見て解るように、上の方はヘッダ処理部のアクセス時間帯である。この時間帯を合わせて、下の白い部分になる。下の色を付いた部分は入出力制御部のアクセス時間帯を表している。

以上ラベルスイッチアーキテクチャの構成と基本処理アルゴリズムを説明した。このようなスイッチが広帯域通信ネットワークの回線速度にどうのような程度で対応できるか、またスイッチにおけるサービスの品質はマルチメディアトラヒックの要求を満たすことができるかどうか等の問題の検討が必要になる。

回線速度の対応能力はスイッチング処理能力、つまり処理のスループット、特にI/O処理能力に関わっている。サービスの品質は主に遅延と廃棄率に関わる。スイッチにおける遅延は主に処理にかかる時間とバッファリングにかかる時間である。バッファリングにおける待ち時間は大きな割合を占めており、回線の使用率と関係がある。廃棄率はバッファサイズと関係がある。次の章でこれらの問題にめぐって検討を進める。

第 4 章

処理スループットの検証

マルチメディア通信に向けたスイッチアーキテクチャを実現するために、スイッチング処理のスループットとスイッチにおける遅延の二つ最も基本的な指標を考えなければいけない。

本研究で提案したスイッチアーキテクチャの実現可能性を明らかにするためにシミュレーションによって基本処理のスループットとスイッチにおける遅延問題を検討した。

新しい発想や提案等が実現できるかどうかを確かめるために、コンピュータシミュレーションによって解答を探すことがより経済的な手段と認められる。シミュレータはシリコンチップとして実現されたものではなく、ソフトウェア的に実現されている。したがって、新しい命令を追加したり、新しいシステムを構築したり、単にデータを収集したりする目的のために、非常に簡単で変更できる。

シミュレーションには幾つかの方法がある。実際に実験をするのが困難なものをモデル化して疑似的に実験をするシミュレーションや、ハードウェアの動作を模倣するシミュレータ上に実際のハードウェア上でも動くようなプログラムを動かすようなシミュレーションなどがある。

本研究においては、基本処理のスループットとスイッチにおける遅延問題にめぐって二つのシミュレーションを行なった。一つは MIPS アーキテクチャの命令セットを実行するプロセッサの動作を模倣したシミュレータ上にスイッチングプログラムを動かして、スイッチング処理に関するシミュレーションを行なった。もう一つは自作シミュレータ上に出力優先制御に関するシミュレーションを行なった。本章では基本処理のスループットに関するシミュレーションを説明する。

4.1 処理スループットの目標

マルチメディア通信に向けるために、スイッチの回線速度は少なくとも 155Mbps が必要である。本研究では、まず 155Mbps の回線速度を狙っている。したがって、入力回線本数 8、回線速度 155Mbps の場合として、次のような見積もりになる。

以下の計算は入出力制御部とヘッダ処理部が同時に並列処理と言う前題条件に基づく想定し、入力回線利用率は 100%という極端的な場合と仮定して、行なうものである。

フレーム長が最大の 5K オクテットの時、1 フレームあたりのスイッチング時間は：

$$\frac{5120[byte/frame] \times 8[bit/byte]}{155 \times 10^6[bit/sec]} = 264258[nsec] \approx 264[msec]$$

この場合、264[msec] の内、入出力制御部は 5[KByte] を入力バッファから、共有メモリに書き込み、5[KByte] を共有メモリから、出力バッファアレーへ読み出し、つまり共有メモリを二回アクセスしなければ行けない。8 本回線で、16 回アクセスになる。この場合、ヘッダ処理部は一番楽で、ただ 8 個ヘッダを処理するだけで済む。

フレームヤード メモリのバンド幅は：

$$\frac{\text{読み書き総バイト数}}{\text{読み書きする時間}} = \frac{5111[byte] \times 2 \times 8[line]}{264258[nsec]} \approx 300[Mbyte/sec]$$

である。

一方、フレーム長が最小の 57 オクテットの場合 1 フレームあたりのスイッチング時間は：

$$\frac{57[byte/frame] \times 8[bit/byte]}{155 \times 10^6[bit/sec]} = 2941.935[nsec] \approx 2942[nsec]$$

である。この時フレームヤード メモリのバンド幅は：

$$\frac{48[byte] \times 2 \times 8[line]}{2942[nsec]} \approx 260[Mbyte/sec]$$

になる。その時共有メモリのバンド幅は

$$\frac{8[byte] \times 4 \times 8[line]}{2942[nsec]} \approx 90[Mbyte/sec]$$

である。

逆に計算すれば、フレーム長が最大の 5K オクテットの場合と同じ時間帯内、フレーム長が最小の 57 オクテットの場合、5120/57=90 個、つまり、264[msec] で 1 回線に 90 個小フレームが到着という意味である。もし最大フレーム長を I/O 処理の単位とすれば、こ

フレーム長	1フレーム当たりの処理時間	I/O プロセッサの仕事	ヘッダ処理プロセッサの仕事
5KByte	264258(nsec)	40KB(5*8) 書き込み 40KB 読み出し	ヘッダ 8 個
57Byte	2942(nsec)	456B(57*8) 書き込み 456B 読み出し	ヘッダ 8 個

表 4.1: スイッチの処理時間の制限

の場合、入出力制御部の仕事はほぼ変わらないが、ヘッダ処理部は一番忙しい時期に直面する。つまり 8 回線合わせて、$90 \times 8 = 720$ 個ヘッダを処理する必要となる。

よって、入出力制御部の処理能力は 264[msec] で、入力 $5[KByte] \times 8[line] = 40[KByte]$（トラヒックがピークの時)、出力 40 [KByte]（常に)。つまり

$$\frac{80[Kbyte]}{264258[nsec]} \approx 300[Mbyte/sec]$$

である。こういう設計をした理由は、入力トラヒックより出力が少し大きい方が交換機内部でフレームが溜らないからである。ちなみに、ここの計算は共有メモリのアクセス権管理問題をまだ考えていない、もし入出力制御部が割り込み処理にかかる時間も考えれば、最悪の場合、つまり入力回線利用率は 100%である上に、入力フレームは全部短いフレームである場合、入出力制御部の処理能力に対する要求がもっと厳しくなる。

一方、ヘッダ処理部の処理能力は：

$$\frac{264258[nsec]}{90 \times 8} \approx 367[nsec]$$

つまり、1 ヘッダ当たりの処理時間は 367[nsec] 以下である。

また、フレームヤード メモリのバンド幅は 300[Mbyte/sec] が必要であり、

共有メモリのバンド幅は 90[Mbyte/sec] が必要である。

そして、処理時間の制限をまとめて表 4.1に示す。

以上述べたスイッチング処理時間の制限に基づいて、基本処理のスループットを検討する。

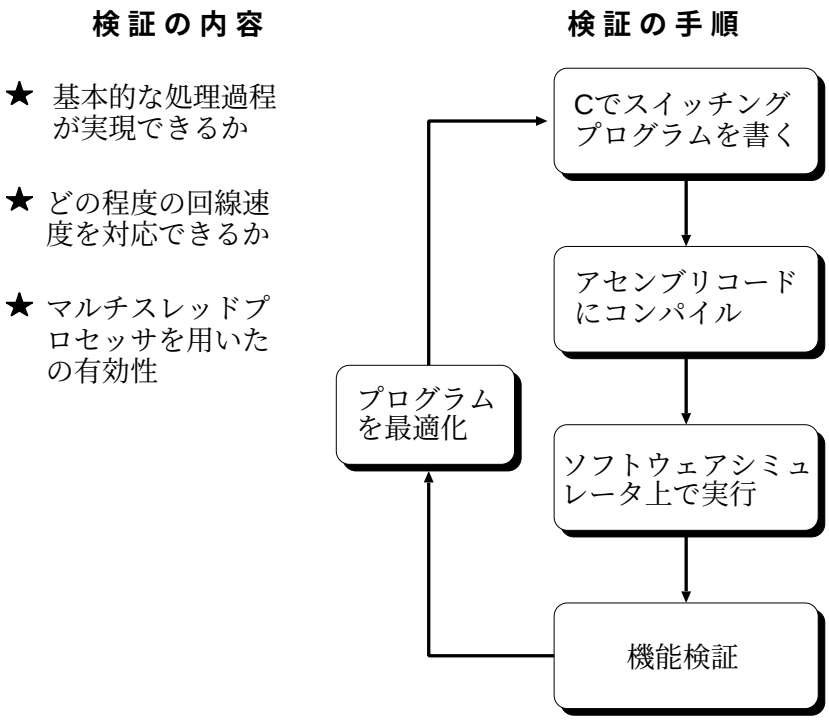


図 4.1: 機能検証の内容と手順

4.2 シミュレーションの内容と方法

スイッチング処理に関するシミュレーションでは、本提案のスイッチにおいて基本処理のスループットを検証し、ソフトウェア処理による可変長フレームスイッチングが実現可能であることを示す。シミュレーションの主な内容と手順を図 4.1で示す。

提案したスイッチでは入出力制御部を汎用プロセッサが担当しており、ヘッダ処理部をマルチスレッドプロセッサが担当している。両方ともにRISC[1]の代表的なMIPS プロセッサだと想定している。そしてこのようなプロセッサをベースにして、ラベルスイッチを構築してみた。基本動作のシミュレーションはMIPS アーキテクチャに向けたソフトウェアシミュレータSPIM[2] 上で行なった。シミュレーションのプログラムのフローチャートを

[1]Reduced Instruction Set Computer の略語、縮小化された命令体系を持つマイクロプロセッサのアーキテクチャを指す。使用頻度の高い命令をできるだけ高速に実現し、その他の命令は実現された命令の組合せとしてプログラムを作成すると、ハードウェアは簡単化されかつ高速化が達成される。1命令1マシンサイクルで動作し、命令の長さの固定である。UNIX オペレーティングシステムを持つ高性能ワークステーションの普及の一因をなす。

[2]SPIM はMIPS R2000/R3000 プロセッサ用に書かれたプログラムを実行するシミュレータである。SPIM の名前は単に MIPS をそのまま逆に綴ったものである。SPIM はアセンブリ言語ファイルを読み込んで直ち

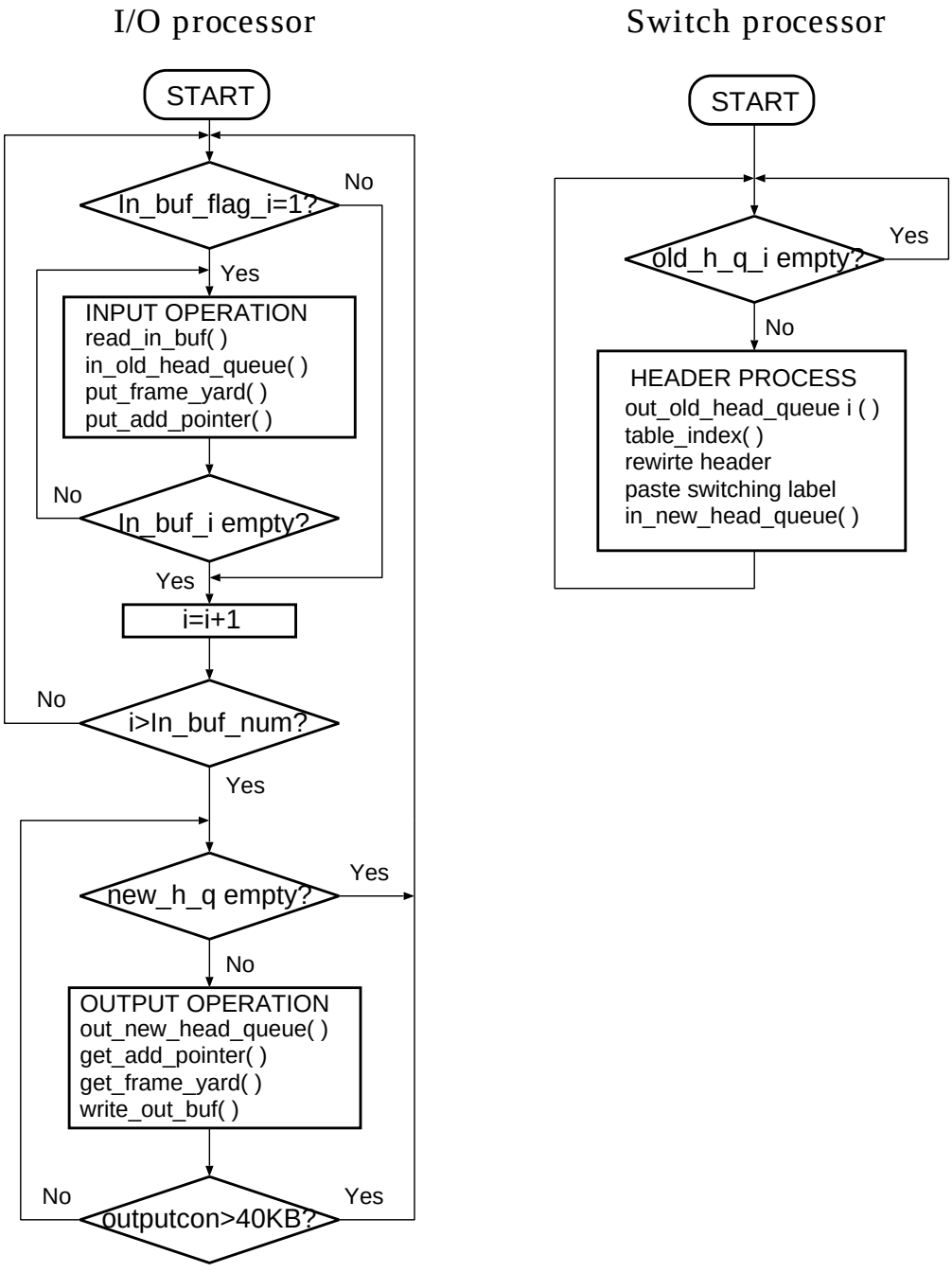


図 4.2: スイッチング処理シミュレーションのプログラム

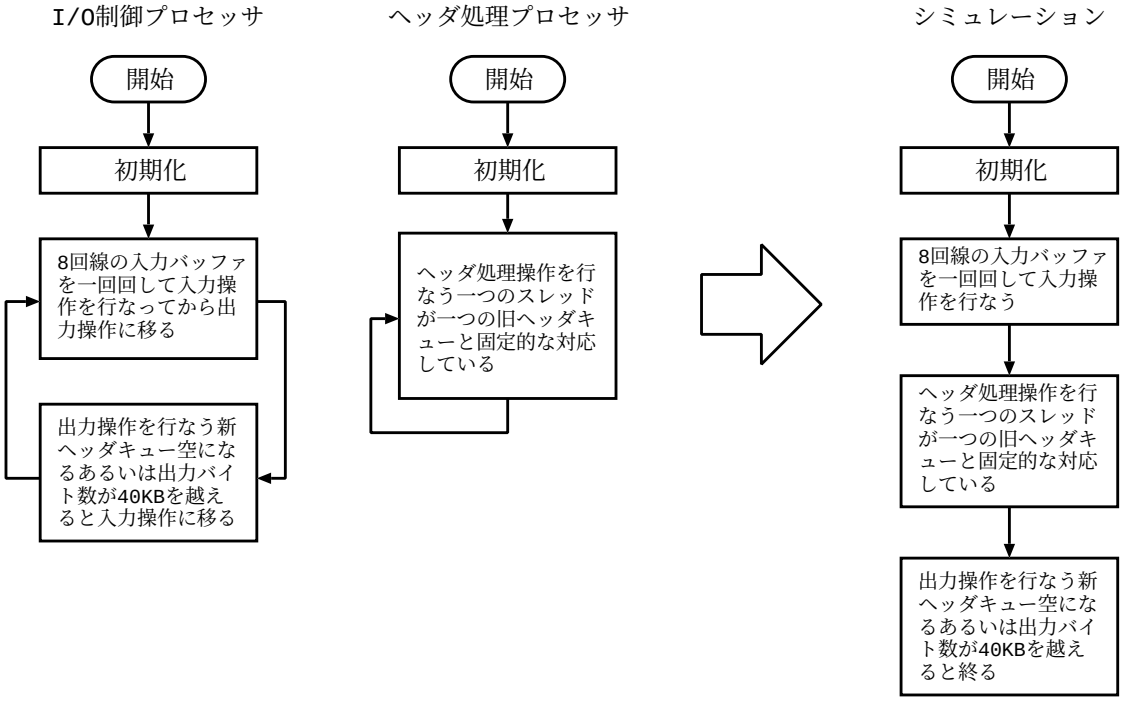


図 4.3: スイッチング処理シミュレーションの方法

図 4.2 で示している。I/O 制御プロセッサの処理は主に入力操作と出力操作二つの部分に分けられ、以下の通りである。

入力操作：

- 入力バッファからフレームを読みだし。
- フレームヘッダ部分を旧ヘッダキューへの書き込み。
- フレームボディー部分をフレームヤードへの書き込み。
- フレームボディーの居る場所を指すポインタをアドレスポインタキューに保存。

出力操作：

- 新ヘッダキューから処理されたヘッダを読みだし。

に実行することができ、特に SPIM には X ウィンドウ用のインタフェースが備わっているので、優れたシミュレーション環境を提供している。SPIM シミュレータはアメリカのウィスコンシン大学マジソン校コンピュータ科学科の James R.Larus 教授に開発され、フリーソフトウェアとして誰でも自由に利用できる。

- フレームボディーの居る場所を指すポインタの取得。
- フレームヤードからフレームボディー部分を読みだし。
- フレームを組み立てて、出力バッファへの書き込み。

ヘッダ処理プロセッサの処理は以下の通りである。

- 旧ヘッダキューからヘッダを読みだし。
- ヘッダ情報にしたがって、ルーティング情報を参照。
- ヘッダ情報の書き換え。
- スイッチラベルをヘッダの先頭に張り付け。
- 処理したヘッダを新ヘッダキューに書き込み。

そのプログラムを NEWS-cc（mips-compiler）というコンパイラを使用して、最適化をするようにコンパイルする。そして、コンパイルされた MIPS のアセンブリコードを SPIM 上で走らせ、それぞれの操作のアセンブラレベルの実行された命令数を計る。

図 4.3では、シミュレーションの方法を簡単に示している。左側は I/O 制御プロセッサとヘッダ処理プロセッサの実際の動作であるが、SPIM 上では一つのプロセッサしかを動作させないので、右側のような方法を採った。

4.3 シミュレーション１：基本処理の検証

シミュレーション１では、単純な方法で基本処理の実行状況を調べる。

1. 方法

   前節で述べたプログラムはまずワークステーション上で実行して、スイッチング処理の動作を確認する。続いて、このプログラムを NEWS-cc により最適化コンパイルして、SPIM 上で実行した。そして実行命令数を計る。さらにコンパイルしたアセンブリコードを一つ一つを追ってみて、レジスタの使用などにおいて、余分なところを除いて、実行命令数を抑えるように人手最適化した。そして実行命令数を計る。これらの実行命令数から８回線合わせて総実行命令数を算出できる。さらに、本仕様のプロセッサを１ GHz で動作する MIPS プロセッサを想定し、パイプライン動作によって、１クロックサイクルで１命令実行と仮定して大まかで計算する。す

フレーム長	１個フレームの処理	実行命令数（最適化コンパイラ）	実行命令数（人手最適化）	８回線総実行命令数
5KByte	入力操作	26989	25703	205624
	ヘッダ処理	148	144	1152
	出力操作	38470	28220	225760
57Byte	入力操作	402	382	3056
	ヘッダ処理	148	144	1152
	出力操作	485	367	2936

表 4.2: シミュレーション１の結果

ると１命令の実行時間は１ ns となる。そして、実行命令数から実行にかかる時間を導出し、その値によってスイッチの処理時間の制限を満たせるかどうかを評価する。

注意として、本来はヘッダ処理プロセッサはマルチスレッドプロセッサなので、同じ動作の複数のスレッドが並列実行という形となるが、この方法でシミュレートするスイッチングプログラムは、他のスレッドが走っている場合を想定していない。あくまで、単独でスレッドプログラムが走っている時のかかる実行命令数を調べて、単純に評価することに主眼を置いている。

2. 結果

表 4.2の結果を得た。

3. 考察

まず、この結果が表 4.1の制限時間と比較する。ヘッダ処理プロセッサは８個ヘッダを処理するのかかる時間が 1152（nsec）なので、制限時間 2942（nsec）の半分もかからず、全部小フレームの一番忙しい場合でも、問題がないと考えられる。しかし、I/O プロセッサは実行時間を大きく越えたことが解った。特に、小フレームの場合

$$3056 + 2936 = 5992\,(nsec) \gg 2942\,(nsec)$$

そこで、I/O プロセッサの処理方式をやり直さなければ行けない。

4.4　シミュレーション２：I/O プロセッサの処理能力評価

シミュレーション２では、I/O プロセッサの処理能力向上にめぐって行なった。I/O プロセッサの処理は非常に単純で、データの転送だけなので、改善するみちはいくつか考えられる。

- I/O プロセッサの制御する回線数を減らす。
- データバスの幅を大きくする。
- もっと速いデータ伝送方式を採る。

そして、I/O プロセッサは MIPS 64bit プロセッサを想定して、シミュレーションを行なった。

次世代高性能 MIPS の土台を象徴すると言われている MIPS64 アーキテクチャは 32bit アーキテクチャの MIPS32 と互換性を持っている。32bit のアドレスイング、64bit のデータオペレーション及び全 64bit のアドレスイングとデータオペレーション両方ともサポートしている。このアーキテクチャの互換性により、MIPS32 のコードはそのまま MIPS64 の上で実行でき、書き直す必要がない。

1. 方法

   シミュレーション１と同じ方法で、ただし I/O プロセッサの書き込みと読みだし幅は 32bit から 64bit に変わった。

2. 結果

   シミュレーション１の結果を表 4.3で示している。

3. 考察

   ヘッダ処理は主にビット操作なので、データバスの幅の増加による影響が全くなかったが、データバスの幅の増加は入出力操作に対してかなり効果がある、特に長いフレームの場合実行命令数はほぼ二分の一になた。フレーム長 5KByte の場合、入出力操作を合わせって８回線で総実行命令数は 216656（nsec）であり、限られた時間の制限を満たすことができた。しかし、フレーム長 57Byte の場合、入出力操作を合わせって８回線で総実行命令数はまだ 3616（nsec）である。限られた時間の制限（2942nsec）を満たすことができなかった。

フレーム長	１個フレームの処理	実行命令数	８回線総実行命令数
5KByte	入力操作	12922	103376
	ヘッダ処理	144	1152
	出力操作	14160	113280
57Byte	入力操作	242	1936
	ヘッダ処理	144	1152
	出力操作	210	1680

表 4.3: シミュレーション２の結果

こういう状態なら、一つI/Oプロセッサは８回線の入出力を制御することが困難であることがわかった。逆に言えば、８回線の入出力を制御するために、二つのI/Oプロセッサが必要と言える。

4.5 シミュレーション３：パイプライン動作の効果

さきの二つのシミュレーションでは実行時間が命令数で計算されてあり、大まかな見積りだと考えられる。シミュレーション３ではパイプライン動作するように想定して、シミュレーションを行なった。

MIPS アーキテクチャでは、殆んどの RISC コンピュータと同じように遅延分岐、遅延ロードを実装している。遅延分岐では、分岐先に行くまで２クロックをかかる。第２クロックでは分岐命令直後にくる命令を実行する。この命令は有用命令あるいは nop 命令である。同じように、遅延ロードも２クロックをかかる。それは、主記憶からロードされた値をロード直後の命令が使用するタイミングを遅らせるためである。

MIPS ではこの複雑さを巧みに隠蔽するために、仮想マシンをアセンブラに導入している。この仮想マシンは実際のハードウェアより遅延分岐、遅延ロードなし、豊富な命令セットを持つというように見える。アセンブラが遅延スロットを満たすように命令を再構成（reorganize：並べ替え）するのである。また、仮想マシンでは、実際の命令を幾つか連ねたものに相当する擬似命令（pseudo instruction）を用意している。

SPIM はこの豊富な命令セットを持つ仮想マシンをシミュレートしている。ちなみに、裸のハードウェアをシミュレートさせることもできる。しかし、SPIM はあくまでもシミュ

レータなので、実際のコンピュータとは異なる点もいくつある。まず、SPIM はキャッシュもしくはメモリのレイテンシーをシミュレートしていない。浮動小数点演算や乗算命令及び割算命令の遅延も正確に反映していない、また、擬似命令はいくつかの実際のマシン命令に展開されるが、遅延スロットを満たすような命令を再構成することは行なっていない[6]。

1. 方法

   パイプライン処理のシミュレーションにおいて、以下のような前題条件をしていた。

   - MIPS 仮想マシン上で行ない、より単純に評価することを望む。
   - SPIM 上では、プロセッサにインターロック機構がない、その上、遅延スロットを満たすような命令を再構成することも行なっていないので、依存関係のある命令間に無効命令（nop) を挿入して遅延を入れる。これは分岐命令とロード命令も含む。
   - MIPS64 の静的分岐予測と投機実行機能を SPIM がシミュレートできないので、しなかった。
   - 本提案の二つプロセッサはハーバードアーキテクチャとする。スイッチングプログラムの規模が小さく、単純な処理なので、高速な小容量メモリに収容可能である。実際に作ったシミュレーションのプログラムを見ると命令メモリは 16KB さえあれば十分だと考えられる。データメモリとはヘッダ処理プロセッサのデータメモリを共有メモリにして、ヘッダキューだけを入れるので、容量も小さい。I/O プロセッサのデータメモリは主にフレームヤードを設けるため、少し大きいが、40KB あれば収容可能である。それによって、二次記憶は必要がないと考えられる。MIPS64 では 16KB 命令キャッシュと 16KB データキャッシュを備えているため、逆に言えば、全てのメモリはキャッシュになれる。そこで、パイプライン処理のシミュレーションでは、キャッシュの動作をシミュレートしなくてもいい、つまりキャッシュミスを考えなくてもいいと認められる。

     実際のシステム構成上は、内部キャッシュへはアクセスできないので、I/O プロセッサとヘッダ処理プロセッサが共通にアクセスできる高速メモリの存在を想定する。

   以上の条件に基づく SPIM 上でスイッチプログラムを走らせ、実行ステップ数を計る。そしてより精確な実行時間を得られる。

フレーム長	１個フレームの処理	実行ステップ数（パイプライン）	４回線総実行ステップ数
5KByte	入力操作	14857	59428
	ヘッダ処理	163	652
	出力操作	16092	64368
57Byte	入力操作	275	1100
	ヘッダ処理	163	652
	出力操作	240	960

表 4.4: シミュレーション３の結果

2. 結果

   実行されたステップ数を表 4.4で示している。

   ちなみに、I/O プロセッサは入出力操作の間に共有メモリをアクセスのステップ数は入力 30、出力 21 であり、合わせて 51 である。これはフレームヘッダの書き込み及び読みだしかかる時間である。フレーム長が 5KByte と 57Byte の二つの場合には同じである。

3. 考察

   この実行した命令ステップ数から、１ GHz で動作する MIPS64 プロセッサにおいて１クロックサイクル１命令のペースで、スイッチング処理のかかる時間をより精確的な計ることができる。

   表 4.4を見ると I/O プロセッサが４回線の処理時間はフレーム長 5KByte の場合、59428+64368=123796(ns)。限られた時間よりまだ 53%の余裕があり、フレーム長 57Byte の場合、1100+960=2060(ns)。限られた時間よりまだ 30% の余裕がある。I/O プロセッサが共有メモリアクセスする際の割り込みオーバヘッドを加えても、この 30%の余りで十分だと考えられる。

   一方ヘッダ処理プロセッサは４回線の処理時間は $(163 + 51) \times 4 = 856(ns)$ である。（51 をプラスする理由は I/O プロセッサは共有メモリをアクセスする間にヘッダ処理プロセッサは割り込みされて、アイドル状態になるからである。）ヘッダ処理プロセッサは限られた時間よりまだ 70%の余裕がある。但し、この場合は、他のスレッドが走っている場合を想定していない、すなわち、ヘッダ処理プロセッサと I/O プ

ロセッサ間の競合はあるが、マルチスレッドプロセッサでは、スレッド間のメモリアクセス競合は生じないである。

以上の結果によれば、二つのI/Oプロセッサが一つのヘッダ処理プロセッサと組合せて、８回線と対応するのは合理的な構成だと考えられる。そうすれば、ヘッダ処理プロセッサの８回線の処理時間が $(163 + 51) \times 8 = 1712(ns)$ で、限られた時間よりまだ40%の余裕がある。

そこで、単にスループットの面から見れば、二つ1GHzで動作する64bitデータバス幅のI/Oプロセッサが、一つ1GHzで動作する32bitのヘッダ処理プロセッサと組み合わせて、８回線のスイッチング処理を行なう場合、本研究提案した処理方式は155Mbpsの回線速度に対応する可能性が十分ある見通しを示すことができた。

4.6 シミュレーション４：マルチスレッド型プロセッサの効果

ヘッダ処理プロセッサの動作は本提案のマルチスレッド方式以外には単一スレッドと複数スレッドで逐次実行の二つの方式が挙げられる。実は本提案における処理方式はマルチスレッド方式しかできない。しかし、比較するために、その二つの方式を処理できると仮定して、処理時間を評価する。

1. 方法

   単一スレッド方式、つまり普通の処理方法で、複数の旧ヘッダキューを順番に回して、ヘッダ処理を行なう。複数スレッド逐次実行方式では、それぞれのスレッドがそれぞれの旧ヘッダキューと一対一対応して、一つのスレッドは一つの旧ヘッダキューしか処理しない。但し全てのスレッドは一つのハードウェアコンテクストを共有して使う。一つのスレッドの処理が終ったら、別のスレッドにコンテクストを渡すという逐次実行方式。一方、マルチスレッド方式と複数スレッド逐次実行方式との区別は複数のハードウェアコンテクストを持つということである、つまり、一つのスレッドは一つのコンテクストを専有している。スイッチング開始する前の初期化の際に、全てのスレッドを一斉立ち挙げて、１命令毎にスレッドを切替え、完全的な並列実行を行なうという方式である。

   このような方法で、シミュレーションプログラムを書き直して、SPIM上で実行させ、得られた実行ステップ数で評価する。

2. 結果

	単一スレッド	複数スレッド逐次実行	マルチスレッド
一つヘッダ処理の平均実行ステップ数	166	225	144

表 4.5: シミュレーション４の結果

得られた結果を表4.5で示す。ここで説明しなければならないことは、ここのマルチスレッド方式の実行ステップ数は 144 である。同じパイプライン実行であるのに、先のシミュレーション３の 163 と等しくない。この原因は SIPM は、マルチスレッドプロセッサをシミュレートすることができないので、先のシミュレーション３のパイプライン実行の際、依存関係がある命令間に無効命令を挿入して遅延を入れることになる。今のシミュレーション４では、マルチスレッドプロセッサと想定して、パイプライン実行の際に、依存関係のある命令がないため、挿入した無効命令を全部取り除いており、シミュレーション２の実行命令数 144 と等しくなる。

3. 考察

マルチスレッド方式と単一スレッド方式と比較すると、１ヘッダ当たりの処理時間を 22ns で短縮でき、約 13%の性能向上が見られた。この原因は二つである、一つはマルチスレッド方式がクロックサイクル毎に依存関係ないのスレッドの命令をパイプラインに投入することで、ハザードの回避ができるので、１クロックサイクル１命令のペースで実行することが可能である。単一スレッド方式では各パイプラインステージに同じスレッドが入っているので、ハザードにより無効命令（nop）の挿入などで実行命令数が増えてしまう。もう一つはマルチスレッド方式では各スレッドが各旧ヘッダキューと一対一対応するため、各スレッドの初期化後（自分の環境を整えた後）、スイッチング処理だけに専念できる。単一スレッド方式では複数の旧ヘッダキューを巡回するのに時間がかかる。

マルチスレッド方式が複数スレッド逐次実行方式と比べると、１ヘッダ当たりの処理時間を 81ns で短縮でき、約 36%の性能向上が見られた。この原因も二つである、一つは上の一つ目の原因と同じである。もう一つは複数スレッド逐次実行方式では一つのヘッダ処理終った度にコンテクストスイッチが起こる、この際全てのレジスタの退避と復元によるオーバーヘッダが大きくなる。ヘッダ処理プログラムはそれほど大きなプログラムではないので、このように大きなオーバーヘッダのある複数

スレッド逐次実行方式はかなり不利だと解った。

以上の結果によると、本提案のヘッダ処理がマルチスレッド方式を採る有効性が見られた。

4.7 ラベルスイッチの設計の例

以上の検討の結果から、本提案のラベルスイッチの一つの設計の例を取り上げる。まず、処理のスループットを守るために、I/O プロセッサは MIPS64 ファミリ中の MIPS64 20Kc プロセッサを選ぶことにする。このプロセッサは６ステージのパイプライン処理であり、スタティック分岐予測と投機実行も行ない、ほとんどの命令が１クロックサイクルで実行され、1000MIPS の性能を持ち、非常に性能高いと考えらる。

DMA コントローラを調べた結果、GT-64120 システムコントローラが注目された。GT-64120 は全ての高性能 64-bit バス MIPS CPU によるシステム構築に対して、シングル・チップ・ソリューションを提供している。特に、DMA 機能において Fly-By という DMA FIFO を通らずに、二つのローカルデータバスに繋がるデバイスの間に直接データを伝送する方式をサポートしている [14]。この機能を利用して、600MByte/sec でのデータ転送できる。本提案の長いフレームの伝送にとって非常に相応しいものと考えられる。

MIPS64 20Kc と GT-64120 の簡単の紹介を付録 A に収める。

図 4.4で示すように、右側はヘッダ処理プロセッサで、左側は GT-64120 と中心に I/O 制御部になる。二つのプロセッサは二つのゲートを通じて、共有メモリにアクセスする。それで、共有メモリのアドレスとは、両側の同じアドレス空間を与えることができる。アクセスの制御は第三章で述べたように割り込みで行なう。

本提案における DMA 伝送は入、出力バッファとフレームヤードの間である。この三つの部分は全部 SDRAM 中に設けて、ローカルデータバスに繋がるにして、Fly-by 機能を利用することにする。そして、幅 64-bit と動作周波数 75MHz で動作させて、600MByte/sec でのデータ伝送が可能である。

図 4.4が原理図であるから、I/O プロセッサを一つしか書かなかった。実際に、本提案では二つの I/O プロセッサと一つのヘッダ処理プロセッサと組み合わせて、８×８回線のスイッチアーキテクチャを適当だと主張する。もう一つの I/O プロセッサの繋がる方法は全く同じであるから、ここで省くことにした。

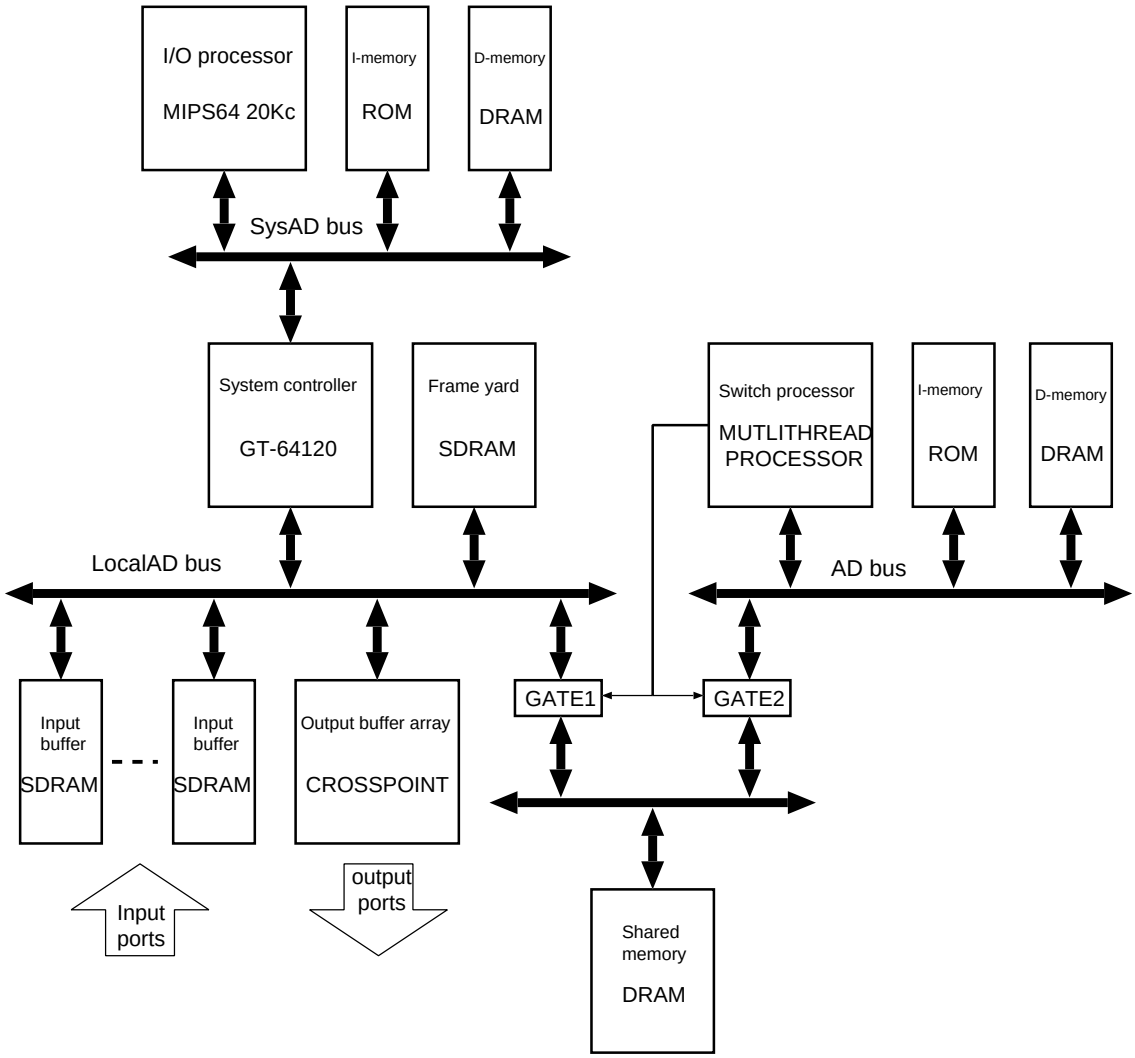


図 4.4: ラベルスイッチの設計の例

第 5 章

入力バッファリングに関する検討

本提案には１回線に対して、入力バッファが二つしか設けないので、あるバッファリング単位の閾値によってバッファの切替えを行なう。それに対して、入出力の処理単位もこの閾値に対応させる。

この閾値が小さいと、場合によって、入力バッファの数が足らない危険がある。この閾値は最大フレーム長と同じにすれば、入力バッファが足らない危険がなくなるが、小フレームに対して入力処理の待ち時間が大きくなる。この閾値を小さくすればするほど入力処理の待ち時間が小さくなる。これは小フレームにとってかなり有利である。ちなみに、小フレームのほうは遅延感受性の高いのトラヒックが多い。しかし、閾値を小さくするのに伴い入力バッファが足らない危険も大きくなる。入力処理の待ち時間の短縮と入力バッファが足らない危険の解消のバランスをとる唯一の道としては、閾値を越える大きなフレームの入力処理時間を、閾値と同じ長さのフレームの入力処理時間と同じ程度に抑えることである。

そういう考え方に基づいて、本提案は以下のような入力バッファリングのポリシを採る。閾値として最大フレーム長と最小フレーム長の間に適当な値を選び、閾値より短いフレームの入力処理を I/O プロセッサ制御で行う。閾値より長いフレームの入力処理をより速いデータ転送方式を検討する。

実際に長いフレームの転送は大きなデータのかたまりの転送だけである。このような対象を高速に処理する方法として DMA[1]方式が有効である。調べた結果、今日の DMA 方式では 600MByte/sec でのデータ伝送が可能だということがわかった。このような DMA 方式を採用すると一つの 5KByte の最大フレームの伝送時間は 8518ns でできる。先のシミュ

[1]DMA という言葉は Direct Memory Access の各頭文字をとった略語であり、I/O とメモリの間あるいはメモリ同士の間のデータ転送を CPU を介さずに、直接的にやってしまう意味で、データ転送の高速化とプログラムの効率化をするために考案された方法である。

レーションの結果によると、この時間はI/Oプロセッサが直接転送で、30個の57Byteの小フレームの転送時間と同じである。つまり1710Byteの転送時間と等しい。そこで、まず入力バッファリングの閾値は1710Byteにする。

5.1 シミュレーション１：異なる入力方式の評価

1. 方法
   具体的には、入力バッファリングでは、一つの入力バッファ中のフレームの長さが合わせって閾値1710Byteを越えると、このバッファのフラッグを立てて、処理のリクエストを示す。そして、次にやってきたフレームから、他の一つバッファに入れ始める。こういうふうにバッファを切替える。一方、I/Oプロセッサは入力回線を順番に回って、フラッグを立てたバッファを処理する。まずフレームの頭を共有メモリの中の旧ヘッダキューに入れて、その際に頭の中のフレーム長のフィールドを見て、長さを知る。もし長さが閾値より小さいならば、I/Oプロセッサが自分でデータ転送を制御する。もし長さが閾値を越えると、DMAに任せる。このバッファが空になったら、次のバッファを処理する。４回線全部終ったら、出力に移る。出力は一回最大4×1710=6840Byteである。出力が終ったら、また入力に移る。このような一回巡回して、88258ns内で終らなければいけない。これは

$$\frac{1710[Byte]}{155[Mbit/sec]} \approx 88258[nsec]$$

   であるからである。

2. 結果
   以上の方法で、シミュレーションを行なった、結果を表5.1でまとめる。

   ここで説明が必要とするのは、まず小フレームの場合、入、出力操作の実行時間はシミュレーション３の結果よりそれぞれ少し大きくなる。原因は先ほど述べたようにI/Oプロセッサがあるフレームをどのように転送するかのを決める前に、まずそのフレーム長を調べなければならない、その分の操作を加えるからである。次に、大フレームの方は、I/Oプロセッサの操作はフレーム長の判断だけではなく、DMA制御レジスタの設定等の操作時間が必要になる。また、SPIMではDMA操作をシミュレートすることができないので、このDMA操作でかかる時間は大フレームのペイロード5111Byteと DMA伝送スピード600MByte/secから算出したものとする。

フレーム長		入力操作 (ns)	出力操作 (ns)	入出力合計 (ns)	４回線合計 (ns)
小フレーム 57Byte	1 個	287	246	533	
	30 個	8610	7380	15990	63960
大フレーム 5KByte	I/O PC 操作	148	98		
	DMA 操作	8518	8518	17282	69128

表 5.1: I/O 操作の時間

3. 考察

表 5.1の結果を見ると、その閾値をさらに短縮することができる。例えば、閾値を 1360Byte にする場合、I/O 処理の単位であるから、要求された処理時間は

$$\frac{1360[Byte] \times 8[bit/Byte]}{155 \times 10^6[bit/sec]} = 70194[nsec]$$

である。小フレームの場合、24 個の 57Byte のフレームを合わせると 1368Byte であり、一つの塊として運ばれる。４回線処理時間は $(287 + 246) \times 24 \times 4 = 51168(ns)$ である。大フレームの場合、1 個の 5KByte フレームを直接運ぶ。処理時間は変わらず、69128(ns) である。両方とも要求された処理時間 70194(ns) を満たすことができる。

よって、本提案では入力バッファリングの閾値は 1360Byte と決めた。そうすると、音声の小フレーム（例えば 48Byte）、MPEG-2 のフレーム（192Byte）等を I/O プロセッサ直接転送という形でする。一方、IEEE802.3 のフレーム（1518Byte）、IEEE802.5 のフレーム（4500Byte）と FDDI のフレーム（4500Byte）など長いフレームは DMA 方式で転送することになる。

5.2 シミュレーション２：入力バッファリングの検討

以上の入力処理方式で、シミュレーションによる入力バッファリングを検討した。

1. 方法

１回線で 155Mbps の入力速度でフレームを到着し、二つのバッファに交替的に入れる、I/O プロセッサはフレームの長さと閾値の比較により二つの方式で入力処理を

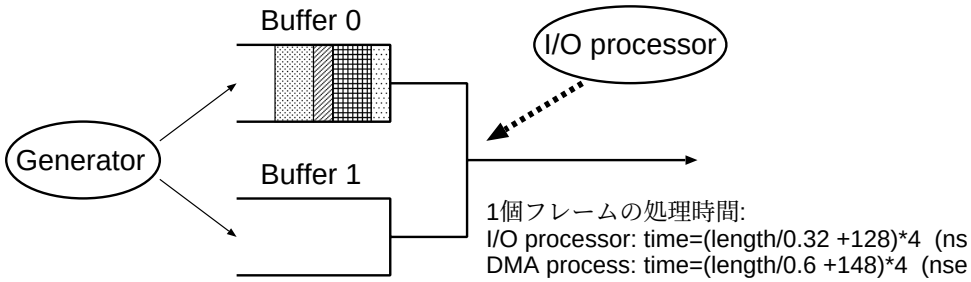


図 5.1: 入力バッファリングのシミュレータモデル

行なう。前章のシミュレーションの結果によって、I/O プロセッサの転送スピードは 320MByte/s である。DMA の転送スピードは 600MByte/s とする。シミュレータモデルを図 5.1で示す。

2. 結果
   二つの入力トラヒックのパターンでシミュレーションを行なった。

   - 場合１
     回線利用率を 100%とする。フレーム長は対数正規分布に従い、平均フレーム長は 4500Byte、1518Byte、192Byte、57Byte とする。４つの平均フレーム長はそれぞれ４分の１の確率を占める。標準偏差は 0.5 である。

     発生間隔はフレーム長と回線利用率に決められる。つまり、１個フレーム発生した後、そのフレーム長と回線利用率によって、次のフレームの発生時間を決める。フレームとフレームの間に間隔時間がない。

   - 場合２
     フレーム長は場合１と同じである。しかし、発生間隔は指数分布に従う。発生間隔の平均値はそれぞれ $929\mu s$、$313\mu s$、$40\mu s$、$12\mu s$ である。そして、平均回線利用率が 100%になる。

   二つの場合の結果を表 5.2と表 5.3でまとめる。一つのバッファの瞬間キュー長の変化を図 5.2、図 5.3で示す。

3. 考察

バッファ	入力した個数	平均キュー長 (Byte)	瞬間最大キュー長 (Byte)	平均待ち時間 (nsec)	最大待ち時間 (nsec)
1	500251	1761	6479	5146	334399
2	499749	1762	6479	4896	334399

表 5.2: 発生 1000000 個、場合１の結果

バッファ	入力した個数	平均キュー長 (Byte)	瞬間最大キュー長 (Byte)	平均待ち時間 (nsec)	最大待ち時間 (nsec)
1	499927	1803	15360	7800	10784364
2	500073	1802	13121	834	11789428

表 5.3: 発生 1000000 個、場合２の結果

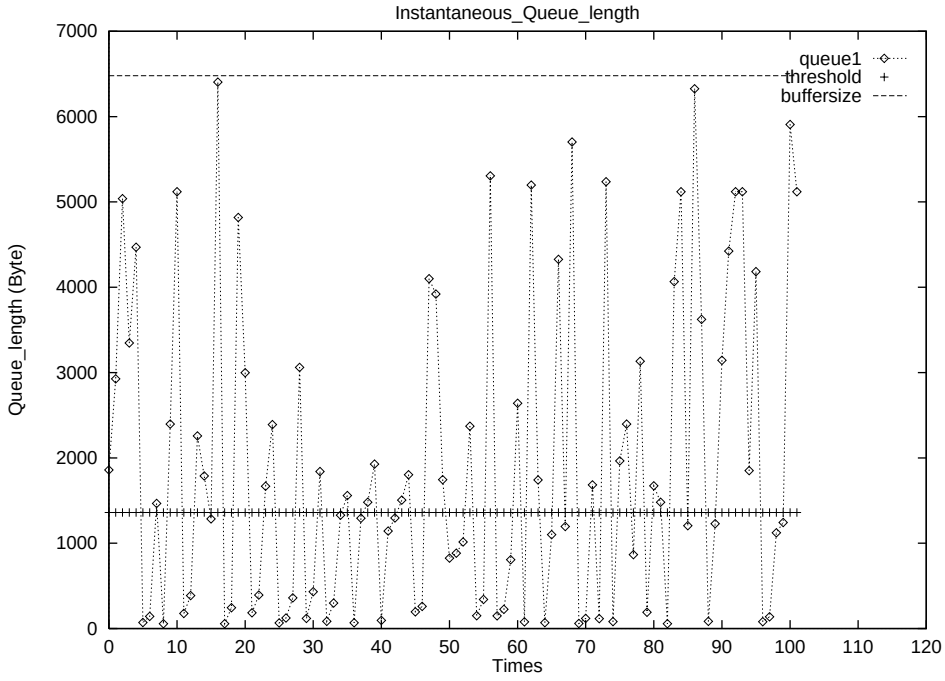


図 5.2: 場合１のバッファの瞬間キュー長

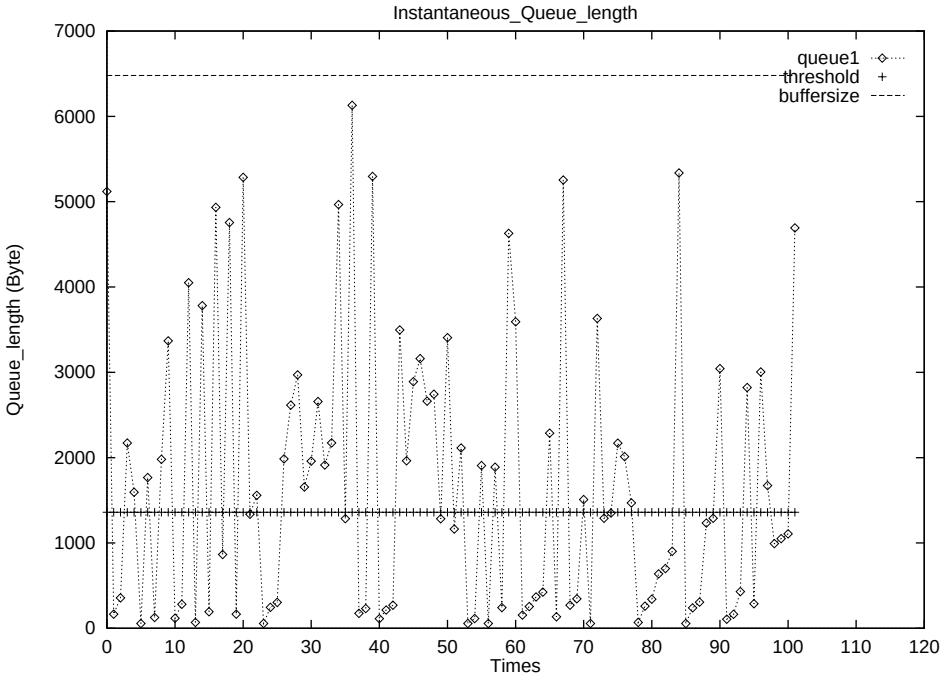


図 5.3: 場合２のバッファの瞬間キュー長

- １）の場合、回線利用率の変動はない、つまりずっと 100%である。最大キュー長は6479Byte であり、これは閾値プラス最大フレーム長（$1360Byte$+$5120Byte = 6480Byte$) より少し小さいである。この結果は予想通りである。
- ２）の場合、回線利用率は変動している。長い時間で統計的な効果では平均利用率は 100%であるが、時々100%を越えることを考えられる。
- 二つの場合ともキューが安定して、平均キュー長 2000Byte 以下であり、平均待ち時間は $10\mu s$ 以下である。I/O プロセッサの能力が十分あることを見える。また、入力バッファのサイズは瞬間最大キュー長によれば、20KByte 以上あれば、入力フレームを蓄えることができる。

第 6 章

出力バッファリングに関する検討

出力バッファリングに関わるシミュレーションでは、提案した出力機構の基本動作、制限時間の役割、バッファサイズ、出力待ち時間等の問題を検討する。

6.1 スイッチにおける QoS 問題

マルチメディア通信に向けたスイッチング方式の検討としては、サービス品質の問題を考えなければいけない。ネットワークにおける品質劣化は主にフレーム誤り、フレーム損失、フレーム混入とフレーム遅延である。本提案の AFFTM 方式において、フレーム誤りは物理レイヤで生じる。フレーム損失は主に物理レイヤの誤り訂正、AFFTM レイヤのノードバッファオーバフロー、AAL レイヤの揺らぎ吸収バッファオーバフロー等により起こる。フレーム混入は物理レイヤと AAL レイヤで生じる。フレーム遅延は主に物理レイヤの伝搬遅延、AFFTM レイヤのノードバッファでの待ち遅延、AAL レイヤのフレーム組み立て遅延と揺らぎ吸収遅延、高層の符号化処理遅延等と考えられる。本研究では、AFFTM レイヤのスイッチング処理なので、AFFTM レイヤにおける QoS 問題だけを検討する。即ち、ノードバッファでの待ち遅延とノードバッファオーバフローによるフレーム損失のことである。

本提案の AFFTM ネットワーク内における伝送遅延は図 6.1のように考えられる。

広帯域ネットワークにおいて、遅延感受性の高いトラフィック（音声、ビデオ）とバーストトラヒックを同時に、かつ物理的に同一のメディア（回線）で伝送する。遅延感受性というのは具体的には、音声トラヒックにおいて ITU ー T で規格化している 25ms から 50ms の遅延、及び 125μs 以下の遅延揺らぎを指す。一方、データトラヒックは遅延の要求が高くないが、損失に対する要求が厳しい。ネットワーク内のフレームの廃棄率はおよ

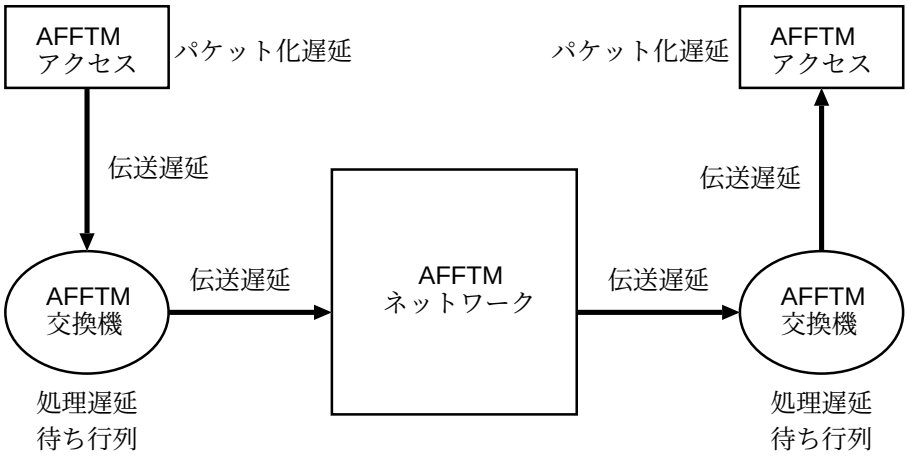


図 6.1: AFFTM ネットワーク内における伝送遅延

そ 10^{-10}、許容値は 10^{-8} と一般的に要求される [11]。

可変長処理では原理上で AFFTM レイヤで遅延揺らぎの制御は不可能である。しかし、現在のネットワークの上位レイヤには大体遅延揺らぎの制御機能を備えているので、(例えば、MPEG-2 の Timestamp 機能など)、下位レイヤでは絶対遅延をある程度で抑えれば、遅延揺らぎの吸収を上位レイヤに任せることが考えられる。つまりリンク層のスイッチングの方針は「Relay as quickly as possible」をとるのは一般的である。

本提案では遅延と関係している主な部分は入力バッファリングと出力バッファリングの二つの部分である。入力バッファリングにおける遅延は前章で検討した。本章では出力バッファリングにおける遅延を検討する。また、本提案の真のバッファリング処理は出力バッファリングだけである、そして、入力バッファリングと共有メモリではスイッチング処理におけるフレーム損失問題がない。フレーム損失に関わるのは出力バッファリングだけである、これも本章で検討する。

6.2 シミュレーションの内容と方法

本提案の出力機構は主に出力バッファアレーと仲裁ロジックから構成される。仲裁ロジックはいくつかのラッチとセレクタから構成され、出力の優先制御を行なう。あるフレームが選択され、そして出力される基準はこのフレームを収納するバッファの優先度とそのフレームの待ち時間である。第三章で述べたように、閾値として、ある制限時間を設

け、出力バッファ内のフレームの待ち時間がこの制限時間を越えると、このバッファの優先度が高くなる。

図6.2の上の部分は４出力回線と４サービスクラスの出力機構を示している。各出力回線の動作が全く同じなので、１回線だけ検討すればよい。いろいろ出力状態を設定しやすいため、図6.2の下の部分のようなシミュレータモデルを決めた。

シミュレーションの内容とは、幾つかの出力パターンにおいて各バッファの中の待ち列長及びバッファ内のフレームがバッファに着いたから、出力されたまでの最大待ち時間を測ることによって、以上述べた問題を検討する。

シミュレーションプログラムのフローチャートを図6.3で示している。

このシミュレータはフレーム発生イベント、出力処理イベントによって駆動するイベント駆動型シミュレータである。次に、ジェネレータチェック、出力実行ユニットチェック、ティック更新、カウンター増し、サンプリング等の構成要素について説明する。

ジェネレータチェック ジェネレータはフレームを発生する時間になるかどうかをチェックする。そうであると、フレーム発生イベントが起こる。(一個フレーム発生して、相応するバッファに入れる。そして決められた平均発生間隔に従って、固定間隔あるいは確率分布による次の発生までの間隔時間を算出する。この時間値は残る時間として記憶される)。そうではないと、ジェネレータの元の残る時間からシミュレーションのステップとしてのティックタイムを減算する。

出力実行ユニットチェック 出力実行ユニットは先の出力操作が終るかどうかをチェックする。そうであると、出力処理イベントが起こる。(最終ラッチからフレーム情報を取り出して、次の出力フレームとする。その際、このフレームの所属バッファ番号と当該フレームの待ち時間がわかる。そして、このフレームの長さと出力回線スピードにより、この出力操作が終るまでの時間を算出する。この時間値は残る時間として記憶される)。そうではないと、出力実行ユニットの元の残る時間からシミュレーションのペースとしてのティックタイムを減算する。ちなみに、最終ラッチからフレーム情報を取り出したら、前の二つのラッチから桁あげを行なう。どちらを選ぶかは優先度と待ち時間によって決められる。図6.2を見るとわかるように、バッファ３の優先度が一番高い。

ティック更新 次のイベントの決定は四つのジェネレータと出力実行ユニットの持つ残り時間が最小のものをもって次のイベントとし、そのイベントに応じた処理を行なう。そしてその最小残り時間を新しいティックタイムとする。このティックタイムはシミュレーションのステップである。

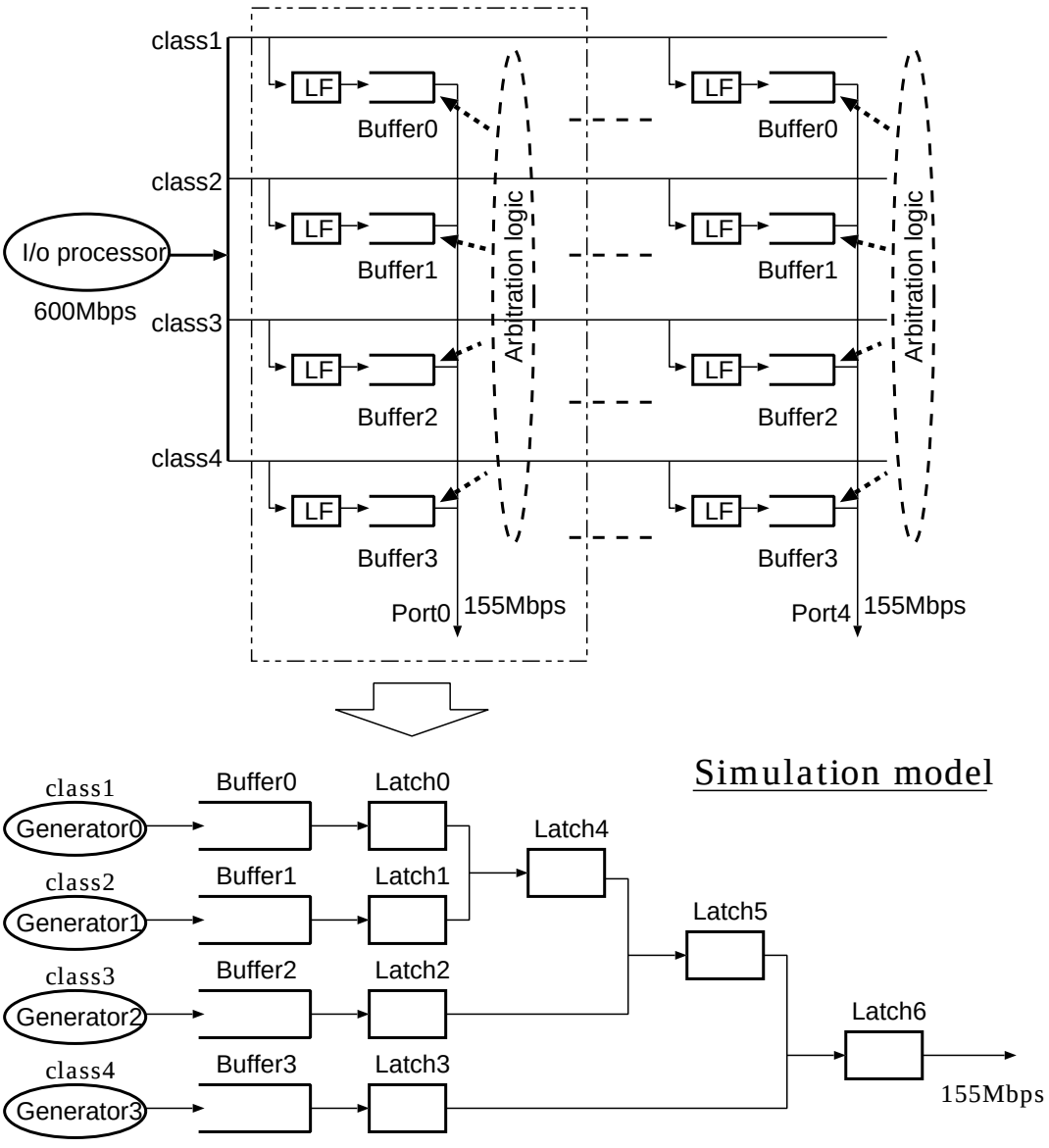


図 6.2: 出力に関するシミュレータモデル

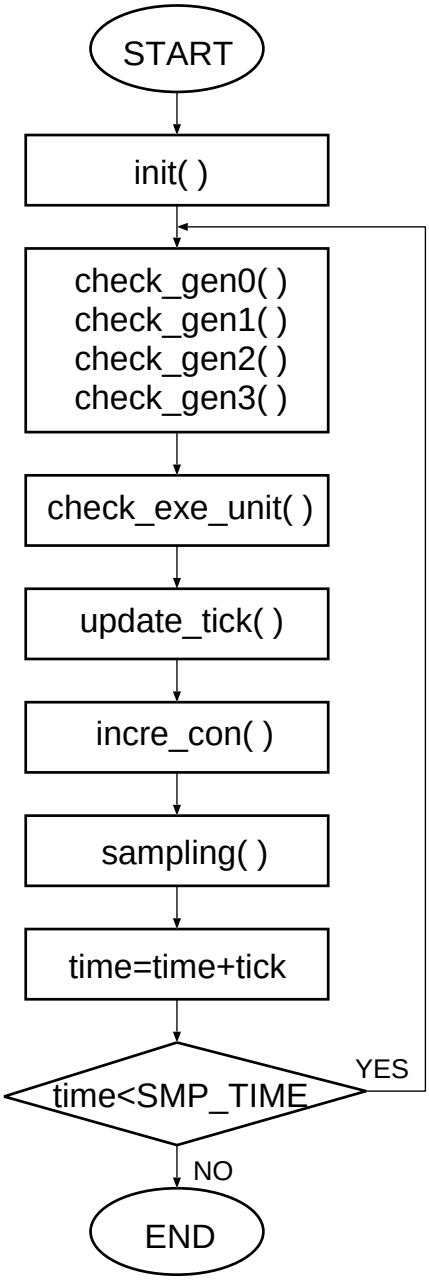


図 6.3: 出力に関するシミュレーションのプログラム

カウンター増し イベントが発生する毎に全てのカウンターにティックタイムをたす。これらのカウンターで全てのフレームの待ち時間を計っている。

サンプリング シミュレーションの全過程において、幾つかのサンプリングウインドウで平均キュー長と瞬間最大キュー長を計る。即ち、ある時間間隔毎にサンプリング開始する、全てのキューのキュー長×ティックタイムで得られる値を足しあげる。同時にティックタイムも足しあげる。サンプリングが終了したら、そのウインドウ内の足しあげたキュー長×時間値を足しあげたティックタイム値で割算して、平均キュー長情報を出力する。

本シミュレーションにおいて、使われる主な術語を説明しておく。

平均キュー長 シミュレーションの全過程において、あるバッファ内の待ち列の平均長さ。Byte で表す[1]。

瞬間最大キュー長 シミュレーションの全過程において、あるバッファ内の瞬間最大待ち列の長さ。Byte で表す。

最大待ち時間 あるバッファ内のフレームの中で、到着から出力されるまで、一番遅いフレームの待ち時間。μsec で表す。

制限時間 仲裁ロジックに用いられた時間閾値。μsec で表す。

回線利用率 出力フレームの平均発生間隔と当該フレームの長さから計算された出力スピードと実際の回線スピードとの比率である。

6.3 シミュレータの動作検証

シミュレーションを行なう前に、構築したシミュレータ自身は想定したとおりの動作を果たすかどうか及び測定値が正しいかどうかを検証することが必要である。

1. 仲裁ロジックの検証

   ４クラス全部同じフレーム長とし、固定時間間隔で発生して、仲裁ロジックの動作を確認する。検証の方法と実験の結果を付録 B に収める。

   その結果から見れば、仲裁ロジックは想定したとおりの動作を果たしていると考えられる。

[1] 通常はキュー長はキューの中の個数で表すが、フレームの長さが異なるので、ここではキューに必要なメモリ量であるバイト数で表す。

クラス	フレーム長（Byte）	平均発生間隔（μsec）	発生間隔分布	相応する利用率
クラス１	4500	1162	指数分布	0.2
クラス２	1518	392	指数分布	0.2
クラス３	192	50	固定間隔	0.2
クラス４	57	15	固定間隔	0.2

表 6.1: パターン１

2. 平均キュー長の検証
   任意の一つのクラスを選択して、このクラスしかフレームをランダム発生する場合、回線利用率と平均キュー長の関係の測定値と M/D/１待ち列モデルの理論値との一致性を検証した。シミュレーションの方法と結果を付録 B に収める。

   その結果によると、測定値と理論値が一致していることがわかった。従って、本シミュレータを用いるシミュレーションで得られる測定値の信頼性があると考えられる。

二つの実験を通じて、シミュレータの基本動作が予想通りだと確認された。また実験の結果の統計的な効果を得られるために、発生フレームが少なくとも 10000 個以上が必要である。

6.4 シミュレーション１：通常の運用状況時の特性

ネットワークは回線速度の約 80%の帯域利用率を目標に設計されるのが一般的な姿である。シミュレーション１では、普通の運用状況と想定して、四つのバッファの平均キュー長と最大待ち時間を調べる。

1. 入力条件
   表 6.1で示すパターン１は入力条件とする。４クラス合わせて利用率は 80%になる。シミュレーション時間は 45sec で、サンプリング間隔は 1sec で、サンプリング時間は 0.5sec とする。

2. 結果
   制限時間は 0、100μsec、500μsec、10000μsec の場合の実験を行なった。それぞれの場合の平均キュー長の変化を図 6.4、図 6.5、図 6.6、図 6.7で示す。制限時間は 0 と

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	38550	396.50	36000(8)	322	6477
クラス２	115165	301.19	12144(8)	232	2034
クラス３	900000	256.35	3456(18)	115	1163
クラス４	3000000	172.71	1710(30)	61	475

表 6.2: 制限時間 0、パターン１の平均キュー長

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	38550	303.00	27000(6)	271	4700
クラス２	115165	292.87	12144(8)	228	1946
クラス３	900000	364.43	4992(26)	145	1355
クラス４	3000000	266.43	3591(63)	86	966

表 6.3: 制限時間 500μsec、パターン１の平均キュー長

制限時間 500μsec の結果データを表 6.2と表 6.3で示す。表の中に瞬間最大キュー長欄の括弧内の数字はフレームの個数である。

以上の結果は全てのクラスは 0.2 の回線利用率を占めしている場合である。続いて、総回線利用率はパターン１と同じくて、各クラス回線利用率は偏りにして、シミュレーションを行なった。その入力条件と結果を付録 C に収める。

さらに、平均回線利用率はパターン１と同じであるが、フレーム長と発生間隔両方共にランダム発生にする。平均フレーム長はパターン１と同じにして、標準偏差は 0.1 と 0.5 の対数正規分布である。発生間隔は指数分布に従い、平均発生間隔はパターン１と同じである。シミュレーションの入力条件と結果を付録 C に収める。

また、フレーム長だけをランダム発生にする、発生間隔はフレーム長とそのクラスの回線利用率によって決める場合のシミュレーションを行なった。平均フレーム長はパターン１と同じであり、標準偏差は 0.1、0.5、1.5 の場合それぞれの結果を付録 C に収める。

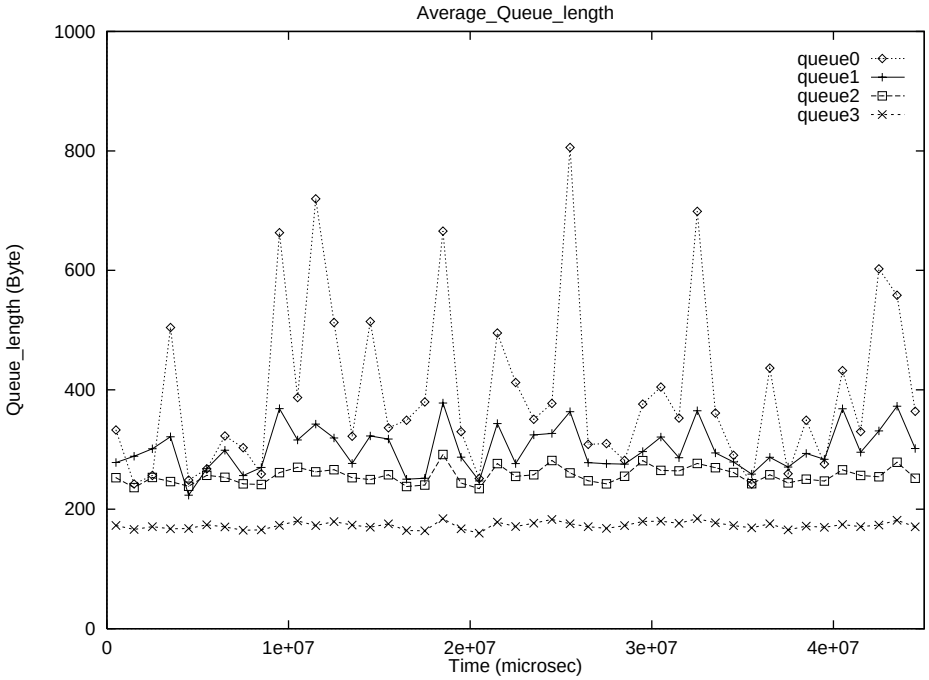


図 6.4: 制限時間 0、パターン１の平均キュー長

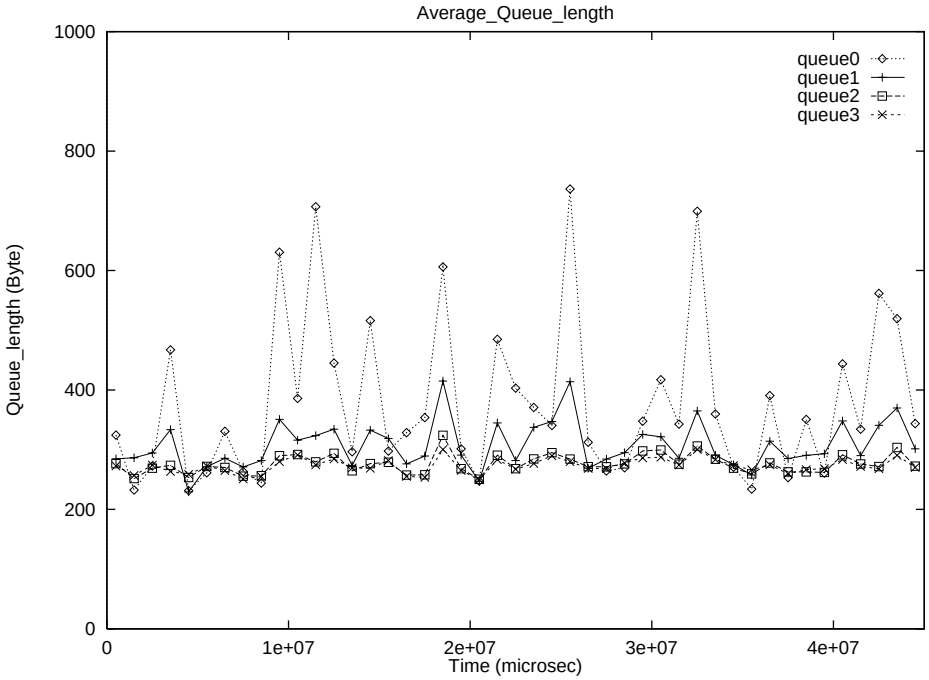


図 6.5: 制限時間 100μsec、パターン１の平均キュー長

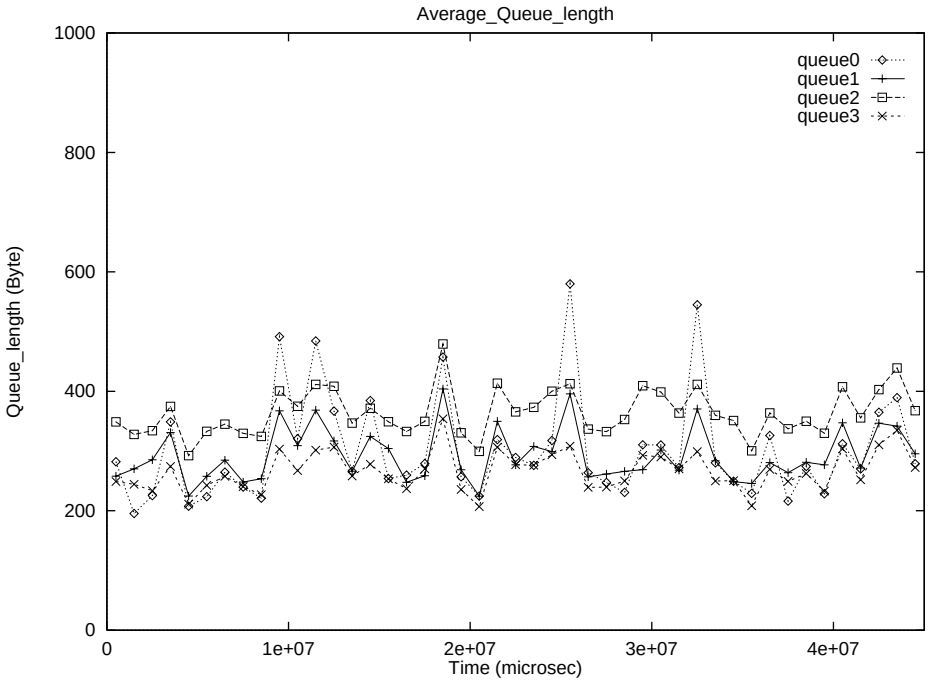


図 6.6: 制限時間 500μsec、パターン１の平均キュー長

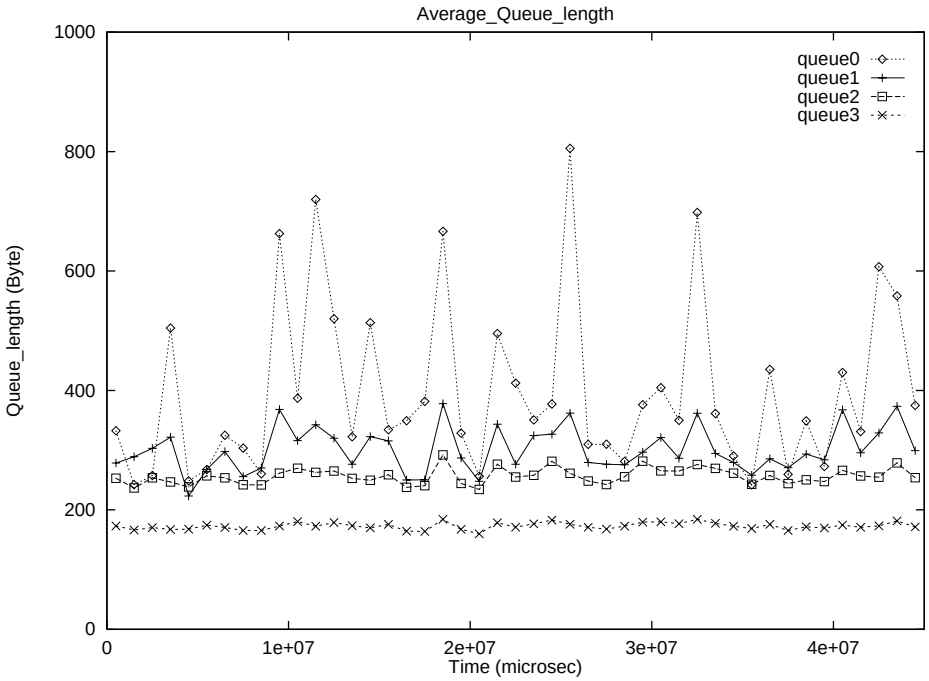


図 6.7: 制限時間 10000μsec、パターン１の平均キュー長

3. 考察

シミュレーション１の結果から、以下のことが解った。

- 回線利用率は 80%の場合、全てのキューが安定して、平均キュー長が小さいである。パターン１の制限時間 0 の場合、つまり完全な優先性による出力の場合、一番優先度高いクラスの平均待ち時間は 61μsec であり、最大待ち時間は 500μsec 位である。一番優先度低いクラスの瞬間最大キュー長が 36KByte 位である。
- 制限時間 0、100μsec、500μsec の図を見ると、制限時間の増大によって、４つのキュー長の差が縮まった。つまり制限時間の役割は公平性を守るということをよく見えた。
- 制限時間 0 と制限時間 10000μsec の図はほぼ同じである、この理由は非常に簡単である。制限時間が小さい過ぎると、全てのキューのフレームの待ち時間が制限時間を越える、制限時間が大き過ぎると、全てのキューのフレームの待ち時間が制限時間を越えることができない、結局両方とも完全な優先性による出力の場合になってしまう。言い換えれば、制限時間が小さい過ぎと大き過ぎの場合、その役割は失ってしまうということである。
- 各クラスの占める回線利用率が偏りする場合、付録 C を参照して、以下のことがわかった。平均キュー長は回線利用率の偏りに従う変化がある。しかし、全てのキューが安定して、平均キュー長は大きくない。また、小フレームは回線利用率を占める割合が大きい場合、全てのクラスの平均待ち時間が小さくなり、大フレームは回線利用率を占める割合が大きい場合、全てのクラスの平均待ち時間が大きくなる。二つの場合は回線利用率同じなので、平均待ち時間が大抵同じはずのに、この結果を出てくる理由はバッファ内の待ち時間が大抵同じであるが、出力を受ける際に、大フレーム自身の出力が時間かかるので、その分の時間が原因になると考える。さらに、回線利用率が偏りする場合は制限時間の役割は弱くなる傾向がある。仲裁ロジックの調停のは、４つのクラスが出力競争する時である。回線利用率が偏りする場合はその出力競争が少ないので、調停の必要がないからである。
- フレーム長と発生間隔と共にランダム発生する場合、パターン１のシミュレーションの結果と比べると、全ての結果を大きくなる。フレーム長分布の標準偏差 0.1 の場合平均キュー長は大抵 46%で増加した。平均待ち時間は大抵 34%で増加した。フレーム長分布の標準偏差 0.5 の場合平均キュー長は大抵 122%で

増加した。平均待ち時間は大抵 78%で増加した。

パターン１のようなフレーム長を固定して、到着はランダムである場合は回線利用率は既に変動している。フレーム長と発生間隔と共にランダム発生する場合では、長い時間の統計的な効果で回線利用率は設定値と一致するが、瞬間の回線利用率はもっと大きく変動している。特にフレーム長は激しく変化する場合。時時設定の回線利用率を大きく越えることを想像できる。

- フレーム長だけランダム発生し、フレームの発生間隔はフレーム長と回線利用率によって決める場合では、平均キュー長は非常に安定して、小さいである。この原因はいくらフレーム長は変動しても、その変動はフレームの発生間隔に見事に吸収されたので、回線利用率の変動は非常に小さくなるからである。フレーム長の分布の標準偏差が 0.5 以下の場合、つまりフレーム長のばらづけが小さい場合、全ての結果はパターン１の結果よりも小さい。

  非同期可変長のフレームに対して、回線利用率の変動を穏やかにするために、それぞれの長さが異なるフレームの間の発生間隔を回線利用率に合わせるように調整すれば、有効なアプローチである。あるスイッチは出力する際に、こういうトラヒック・シェイピングをすれば、続いてのスイッチのバッファリング等の処理にとって、非常に良いである。

6.5　シミュレーション２：回線利用率に対するキュー長の状況

シミュレーション２では、回線利用率と平均キュー長の関係を調べる。

1. 入力条件

   パターン２は入力条件とする。パターン２とパターン１ほぼ同じである、ただ一つの区別は、全ての発生間隔は指数分布に従う。分析を簡単にするために、制限時間が 0 とする。回線利用率を 80%から 98%までと変化させ、平均キュー長を調べる。表 6.4で示しているのは回線利用率 80%の場合である、各々のクラスの平均発生間隔を変わらせて、回線利用率を調整する。シミュレーション時間は 40sec くらいで、サンプリング間隔は 1sec で、サンプリング時間は 0.5sec とする。

2. 結果

   回線利用率 85%、95%と 98%の平均キュー長の様子を図 6.8、図 6.9、図 6.10で示す。回線利用率 85%、95%の結果データを表 6.5と表 6.6で示す。

クラス	フレーム長 (Byte)	平均発生間隔 (μsec)	発生間隔分布	相応する利用率
クラス１	4500	1162	指数分布	0.2
クラス２	1518	392	指数分布	0.2
クラス３	192	50	指数分布	0.2
クラス４	57	15	指数分布	0.2

表 6.4: パターン２

クラス	発生個数	平均キュー長 (Byte)	瞬間最大キュー長 (Byte)	最大待ち時間 (μsec)
クラス１	36933	816.25	40500(9)	7954
クラス２	109972	530.58	12144(8)	2376
クラス３	871948	465.16	7872(41)	1317
クラス４	3000000	254.16	3933(69)	701

表 6.5: 回線利用率 85%、パターン２の平均キュー長

クラス	発生個数	平均キュー長 (Byte)	瞬間最大キュー長 (Byte)	最大待ち時間 (μsec)
クラス１	38578	7884.35	130500(29)	25967
クラス２	114202	1624.63	16698(11)	3203
クラス３	903070	844.88	6720(35)	1320
クラス４	3000000	354.98	3648(64)	702

表 6.6: 回線利用率 95%、パターン２の平均キュー長

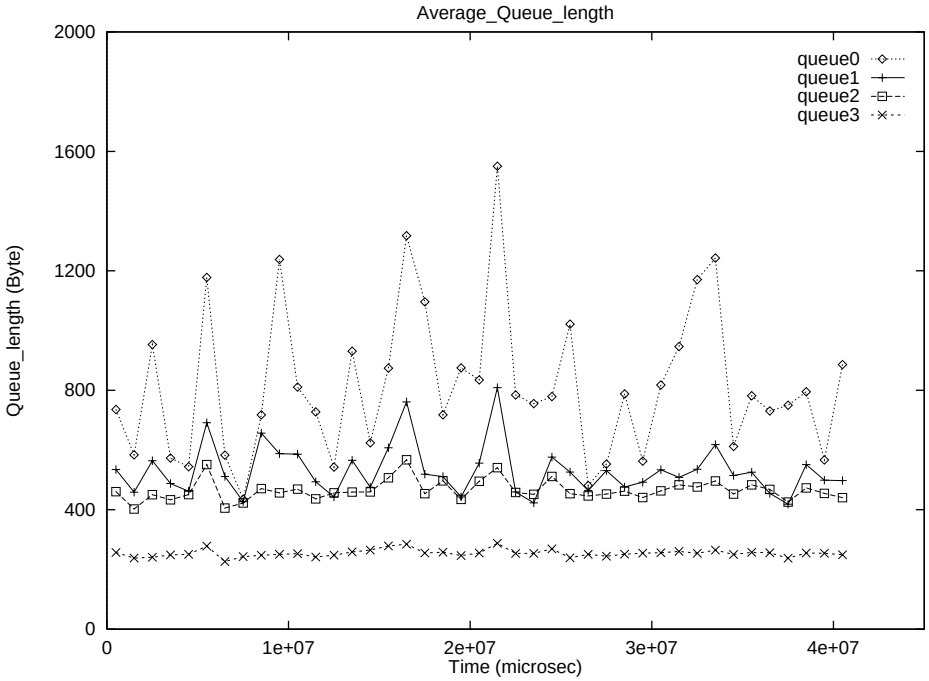


図 6.8: 回線利用率 85%、パターン 2 の平均キュー長

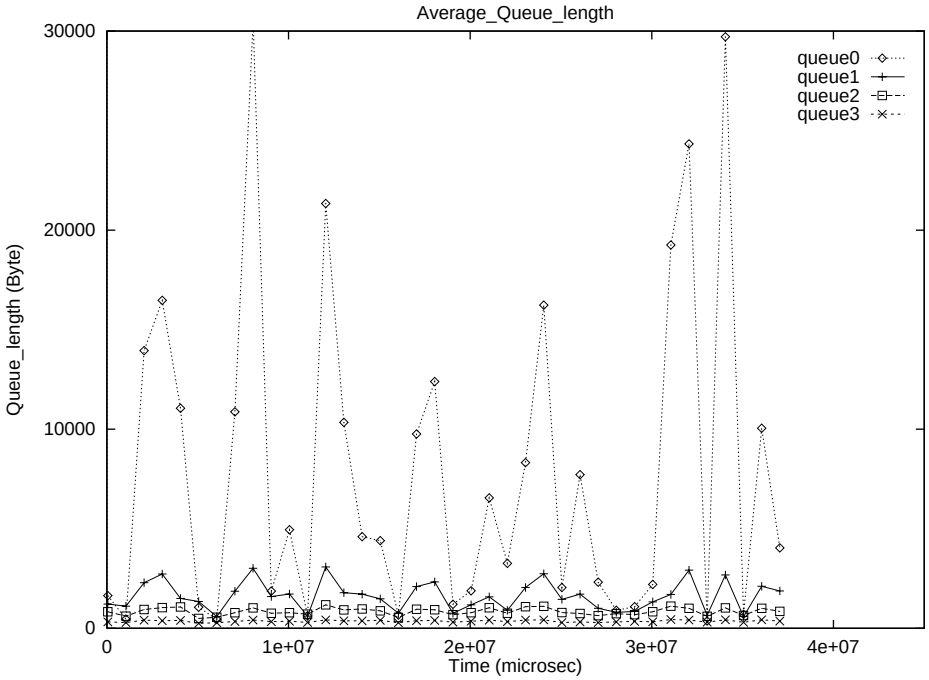


図 6.9: 回線利用率 95%、パターン 2 の平均キュー長

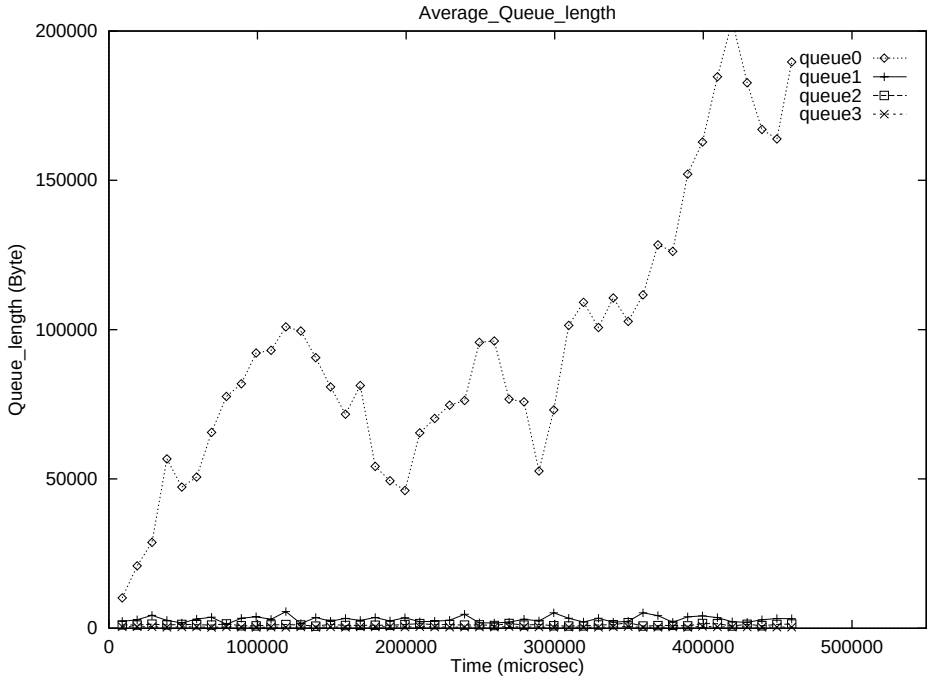


図 6.10: 回線利用率 98%、パターン２の平均キュー長

3. 考察

シミュレーション２を通じて、以下のことが解った。

- 回線利用率は 95%が限界であることが明らかにした。利用率は 95%を越えると、キューの状態が不安定になって、図 6.10で示した利用率 98% のような、平均キュー長が無限に増加する一方になった。バッファサイズを限られたので、この場合フレームを廃棄することがやむを得なくなる。
- 回線利用率は 95%の時、一番優先度高いクラスの最大待ち時間は 700μsec 位であり、一番優先度低いクラスの瞬間最大キュー長が 130KByte 位である。
- 表 6.5と表 6.6から見ると、回線利用率の増加はクラス１に対する影響は大き、クラス３、４に対する影響は非常に小さい。これは優先性の力を見えた。特に回線利用率は限界を越えると、例えば利用率 98%の場合、クラス２、３、４のキューは安定して、クラス１のキューだけ急激に大きくなるということが解った。回線利用率の変動は出力優先制御に対する影響が小さい、制限時間の調整しか出力優先制御に対する影響を与えない、これは望ましい結果である。

- このシミュレーションでは制限時間を 0 としていたので、全く優先性によって出力を行なったということで、先程述べた現象が目立ちである。しかし、回線利用率は限界を越える場合、制限時間を加えても、役立たない。なぜなら、回線利用率は限界を越えるということはこの出力優先制御機構がこんな激しい到着フレームに対応することができないと意味している。この場合もし制限時間を加えると、この制限時間の調整によって、しばらくある程度の公平性を守ることができるが、全てのキューは急成長になる。ある時間を経って、全てのキューは制限時間を越える。そうすると、全く優先性による出力の状態に戻る。ただ、制限時間を加えない場合との区別は、その時全てのキューはある程度のフレームが溜っているだけである。

  この問題を良く考えれば、これこそ制限時間を設けることによって望んでいることである。つまり、回線利用率低い時、優先性と公平性の間に調整する。回線利用率が非常に高い時、制限時間の役割を自動的に失って、優先度高いのクラスを優先出力させるということである。

6.6 シミュレーション３：制限時間とキュー長及び待ち時間の関係

シミュレーション３では、最大回線利用率の場合、制限時間とキュー長及び最大待ち時間の関係を調べる。

1. 入力条件
   入力条件はパターン２と同じである、つまり４クラス全部ランダムで到着。発生間隔は指数分布に従う。回線利用率が 95% と固定して、各クラスの利用率が平均して、0.24%位である。制限時間は 0 から 500μsec づつ変化させ、各バッファの瞬間最大キュー長と最大待ち時間を調べる。サンプリング間隔は 1sec で、サンプリング時間は 0.5sec とする。

2. 結果
   最大待ち時間と瞬間最大キュー長を図 6.11と図 6.12で示す。データを表 6.7でまとめる。

3. 考察
   シミュレーション３を通じて、以下のことが解った。

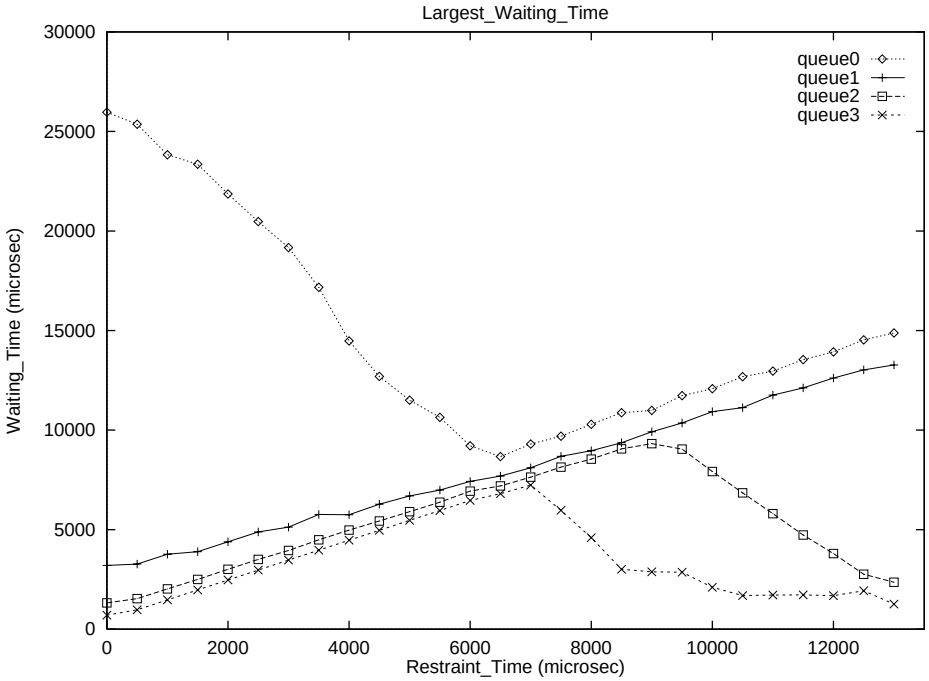


図 6.11: 最大待ち時間と制限時間の関係

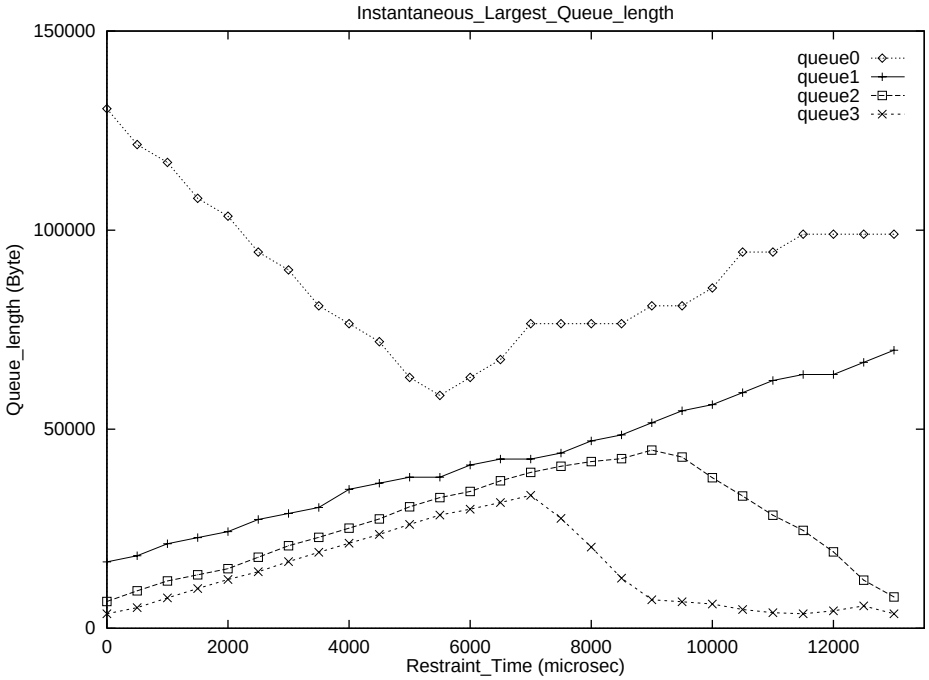


図 6.12: 瞬間最大キュー長と制限時間の関係

制限時間 (μsec)	クラス	発生個数	瞬間最大キュー長 (Byte)	最大待ち時間 (μsec)
1000	クラス１	38578	117000	23827
	クラス２	114202	21252	3765
	クラス３	903070	11904	2023
	クラス４	3000000	7638	1473
0	クラス１	1000000000	301500(67)	60297
6000	クラス２	1000000000	57684(38)	8368
	クラス３	1000000000	38976(203)	7023
	クラス４	1000000000	32889(577)	6471

表 6.7: 利用率 95%の最大瞬間キュー長

- 図 6.11と図 6.12を見ると、その制限時間の役割がさらに明らかにした。先ほど述べたように回線利用率がある限界を越えると、制限時間の役割が自動的失うことが解った。今回のシミュレーションでは、その回線利用率が限界を越えない場合でも、制限時間の限界もある。つまり、制限時間の設定は限度がある。ある値を越えると役割も失ってしまう。今回のシミュレーションの場合では、この値は大体 6000μsec である。実にその理由は同じである。制限時間を 0 とする時、完全の優先性により出力、制限時間を段々大きくすると、公平性が高くなる。ある値（ここは 6000μsec）を越えると、全てのキューの待ち時間が制限時間を越えることができなくなる。かえて優先性の方が強くなる。そこで、制限時間が 0 の時優先性を一番強調し、制限時間が限界値の時公平性を一番強調すると考えてよい。
- このシミュレーションで制限時間の設定は 0 から 6000μsec の範囲内にすべきであることがわかった。但し、制限時間を 6000μsec にすると、全てのキューの最大待ち時間が 6000μsec 位になる。これは遅延要求が高いのクラス４に対して、かなり大きすぎである。そこで、制限時間を 1000μsec 以下に抑えるが適当と考えられる。そうすると、クラス４の最大待ち時間が 1500μsec 以下であり、クラス３の最大待ち時間が 2000μsec 位である。その際、クラス１の瞬間最大キュー長は 117000Byte である。これはバッファ0 のサイズが 117KByte 以上であれば、フレームを廃棄せずに収容できると意味する。

- バッファサイズが限られたため、バッファオーバーフローの場合フレームを廃棄することを考えなけらばならない。これはQoS制御のもう一つ重要な仕事である。廃棄する方法では出力バッファに閾値を設け、貯められたフレームの長さがこの閾値を越えると、廃棄優先度高いのフレームを廃棄する。廃棄操作は貯められたフレームの待ち列長が閾値より小さくまでずっと続けている。廃棄率とバッファサイズの関係を調べるシミュレーションは巨大な数のフレームを発生する必要があるから時間がかかる。本研究ではこのパターン２しか検討しなかった。表6.7の下の部分を見ると解るように、利用率95%制限時間0の場合、バッファ0は一番長いと考えられる。その瞬間最大キュー長は301500Byteである、そして、フレームを廃棄する閾値は300KByteにする。バッファ0のサイズをその閾値の1.5倍にする。即ち、450KByteである。発生個数は1000000000であるので、逆に言えば、バッファ0のサイズを300KByte以上であれば、廃棄率10^{-9}を守ることが可能である。同じように、バッファ1、2、3は利用率95%制限時間6000μsecの場合一番長いと考えられる。その瞬間最大キュー長はそれぞれ57684Byte、38976Byte、32889Byteであるので、それぞれフレームを廃棄する閾値は60KByte、40KByte、32Kbyteにする。バッファサイズをその閾値の1.5倍にすると、90KByte、60KByte、48KByteである。

ここで、注意して欲しいのは最大待ち時間が、シミュレーションの全過程で一番遅く出力されたフレームの待ち時間であるから、全てのフレームの平均待ち時間はこれよりはるかに小さいと考えてよい。

6.7　シミュレーション４：バーストトラヒック時のふるまい

シミュレーション４ではクラス１はバーストトラヒックが発生する場合キューの様子を調べる。

1. 入力条件
   入力条件はパターン１とほぼ同じである。クラス2、3、4を合わせて、87.6% の回線使用率を占める。シミュレーションの時間は30secである。クラス１は5secから20secまでの間にフレームを発生する。発生する時クラス１の使用率は10%とし、その際4クラス合わせて97.6%の利用率になる。そして制限時間が0と500μsecの場合、全てのバッファの平均キュー長を計る。

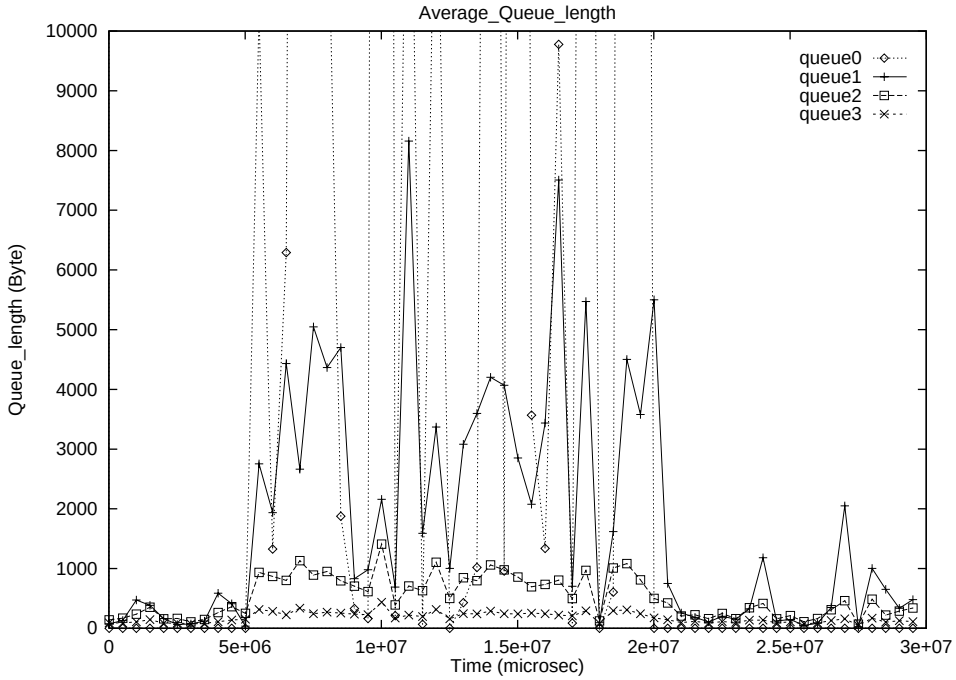


図 6.13: クラス１バースト発生、制限時間 0

2. 結果

   結果を図 6.13と図 6.14で示す。

3. 考察

   - まず図 6.13を見る、これは制限時間 0 の場合。最初キュー 1、2、3 は全部小さく、キュー 0 は 0 である。クラス 1 が発生した時、主にキュー 1 に対する影響を与える。キュー 3 に対する影響が非常に小さい。
   - 図 6.14を見ると、制限時間 500μsec の場合、クラス 1 が発生した時、他の三つのキューに全部影響が与える、キュー 3 の最大平均長さは 3000Byte 位に、キュー 2 の最大平均長さは 3500Byte 位になる。
   - シミュレーションの結果によって、クラス１はバーストトラヒックが発生する場合、優先度の力の原因で、回線利用率は 97.6%であっても、優先度高いのクラス 4 に対して影響が少ないとわかった。しかし、500μsec の制限時間をかかる場合、クラス１のバーストトラヒックによりやってくる襲撃を他の三つのキューに分担する形になる。その際三つのキュー全部長くなって、変動も大きくなる。その平均キュー長の増加の量は総回線利用率に関わっている。

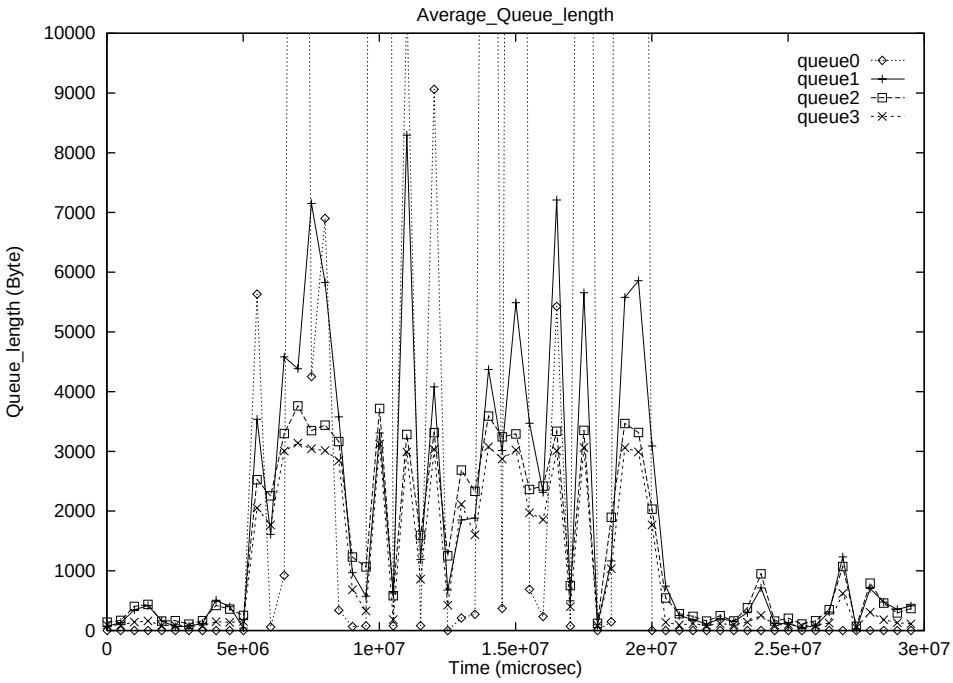


図 6.14: クラス１バースト発生、制限時間 500μsec

第 7 章

結論

本研究では、AFFTM 方式を提案し、AFFTM 可変長フレームを扱うラベルスイッチアーキテクチャを検討した。これらの問題についてまとめる。

7.1 AFFTM 方式のまとめ

AFFTM 方式の提案は可変長フレームを対象とするのに、新しい形式やシグナリング等を起こさずに既存のものの組合せで実現するという考え方に基づき考えたものである。従って、AFFTM 方式の機能は殆んど ATM 方式から継承したものを想定したものである。AFFTM レイヤも ATM レイヤと対応している。しかし、固定長のセルを扱う代わりに、可変長フレームを扱うことになるため、相応した機能がある程度変わらなければならない。本研究では AFFTM フレームフォーマットを提案して、可変長データに対応するための形とした。これにより多様なトラヒックに対してより柔軟な対応をすることができる。特に、長いフレームの場合は伝送オーバヘッドが小さく、スイッチノードにおけるヘッダ処理の量も大幅に削減される。データパケットに対してはるかに有利である。さらに、AFFTM フレーム毎の QoS 制御機能を強化した。これにより高位層からリンクを確立した後、コネクションあるいはコネクションレスにも関わらず、利用可能な QoS 制御機能を提供している。

本研究では、AFFTM 方式の検討を AFFTM レイヤについてだけ行なった。

7.2 ラベルスイッチアーキテクチャのまとめ

本研究の主題として、可変長フレームを扱うラベルスイッチを提案した、提案したスイッチアーキテクチャの特徴を以下にまとめる。

- 出力バッファと共有メモリ型のスイッチアーキテクチャの結合によって内部衝突が生じない理想的なスイッチができる。
- ソフトウェアスイッチング処理とハードウェア制御のクロスポイント出力バッファアレーとの結合によってソフトウェア処理の割合を低減することができる。
- マルチスレッド型プロセッサを用いることにより、ソフトウェア処理の高いスループットを期待することができるだけではなく、全ての入力回線からの長さが異なるフレームを一つの単純な循環バッファ方式の共有メモリに収容できる。また、ヘッダキューをスレッド対応に設置することにより、スレッドの間に負荷のバランスも良くなる。
- ラベル技術を活用して、出力優先制御をハードウェアに任せることにより、QoS 制御が簡単かつ高速で行なわれる。
- 入力バッファリングにおいて閾値を設け、長さが異なるフレームを異なる方法で処理し、入力時間差を抑えることによって入力待ち時間を短縮できる。
- 出力ッファリングにおける仲裁ロジックにより優先性と公平性を調整することができる。
- 提案したスイッチアーキテクチャでは長さが異なるフレームを処理するスイッチアルゴリズムは非常に単純であり、バッファの数量も大きくない。ハードウェアとソフトウェアの両方とも実現しやすい構造であると考えられる。

7.3 機能検証のまとめ

提案したラベルスイッチアーキテクチャの機能検証するために、シミュレーションを行なった。その結果をまとめる。

- 提案したラベルスイッチアーキテクチャを 64bit バス幅と 1000MIPS の性能を持つプロセッサが I/O プロセッサとして、32bit バス幅のマルチスレッドプロセッサが

ヘッダ処理プロセッサとして構成するば、得られた処理スループットは155Mbpsの回線速度に対応することができる。つまりB-ISDNの基本インタフェースの要求を満たすことが可能である。

- 回線利用率が80%という通常の運用状態の場合、出力制限時間を500μsとして、優先度の高いの小フレームのスイッチにおける遅延は、入力待ち時間、処理時間、出力待ち時間を合わせて、最大遅延は$334\mu s + 533ns + 966\mu s \approx 1.3ms$位であり、平均遅延は$5\mu s + 533ns + 86\mu s \approx 0.09ms$位である。これによって、広帯域ネットワークにおける遅延の要求を満たすことも可能である。

- マルチスレッドプロセッサを用いることにより、パイプライン実行の際のハザードの回避ができ、１クロックサイクル１命令のペースで実行することが可能であるので、単一スレッド方式と複数スレッド逐次実行方式よりそれぞれ13%と36%の性能向上が得られる。

- 入力バッファリングにおいて、処理の塊を1360Byteと決めた。I/Oプロセッサの処理能力が十分であるので、全ての入力バッファサイズが$1360 + 5120 = 6480Byte$よりさえ大きくすれば良いだとわかった。また、回線利用率の変動を考えて、入力バッファサイズを20KByteとすれば、十分であると考える。

- フレームヤードのサイズは４回線に一回入力されたフレーム量より多ければ良いと考えられる。$1360 \times 4 = 5440Byte$であるから、２倍の余裕を持たせて、16KByteであれば十分だと認められる。

- 出力バッファリングにおいて、４クラスの使用率が等しくて、合わせて95%の回線利用率を占める場合、クラス1、2、3、4の瞬間最大キュー長がそれぞれ300KByte、57KByte、38KByte、32Kbyteであるので、フレームを廃棄する閾値をそれぞれ300KByte、60KByte、40KByte、32KByteにする。そして、バッファサイズをそれぞれ450KByte、90KByte、60KByte、48KByteにすれば良いとわかった。

- 出力バッファリングに関するシミュレーションから、制限時間の役割が良くわかった。その制限時間の設定によって、優先性と公平性を調整することができる。さらに言えば、高い優先度のクラスの待ち時間と低い優先度のクラスのバッファサイズ（これはフレームの廃棄率と関わっている）とのトレードオフの問題である。

7.4 本研究に関する問題点と今後の課題

以下の問題は今後の課題になる。

- 本研究の主題は可変長の通信方式全体の検討ではないので、今回 AFFTM 方式の検討は AFFTM レイヤしか行わなかった。他のレイヤの機能をどういうふうに実現するかは検討していない。一つの方式全体の検討としてははるかに不十分である。

- 非同期可変長方式はスイッチ処理の複雑さにより処理スピードの面で固定長方式には及ばないが、多種多様のマルチメディアトラヒックに対してより柔軟な対応ができるので魅力もある。ソフトウェア処理に頼らなければならない可変長フレームのスイッチング処理は、今後どのようにして、処理アルゴリズムをさらに単純化し、ソフトウェア処理の割合を抑えるのかが重要な課題と考えている。

- 本研究で提案したスイッチアーキテクチャはスイッチユニットである。本格的な交換機にはまだ成れない。このようなスイッチユニットに物理レイヤハードウェアと合わせて、シグナル制御ユニットを加えて、完全な交換機に構成できる。さらに多数の回線に対応するために、これらのスイッチユニットを多段相互結合網 MIN(Multistage Interconnection Network) で繋ぐことにより、現実の交換機ができあがる。本研究の内容はそれらの部分の中のほんの小さい一部分だけである。提案したスイッチアーキテクチャで本格的な交換機を実現するまでには、まだ沢山の課題が残っている。

謝辞

貴重な留学の機会を作って下さいました、本研究を進めるにあたり、終始熱心かつ寛容な御指導を賜わりました、日比野靖教授に心から御礼申し上げます。

また、適切な御指導、御助言をして頂きました丹康雄助教授、宮崎純助手、落水浩一郎教授、堀口進教授、中島和夫助教授、篠田陽一助教授に深く感謝致します。

さらに、コンピュータ環境に関する沢山の知識を教えて頂きました、池田吉朗君と鵜飼和歳君に感謝致します。

その他、貴重な御意見、御討論を頂きました日比野研究室の皆様を始め、多くの方々の御助言に対し厚く御礼申し上げます。

長い間に、沢山の日本語を教えて下さいました、堀口悦子先生と鎌田倫子先生に深く感謝致します。

また、いつも励まして下さいました、北陸カラーフォームの中山知英氏に心から感謝致します。

最後に、私の快く大学院の留学生活を暖かく支援下さいました、アイ・社会文化推進事業団に深い感謝の意を表します。

2000 年 早春

計算機アーキテクチャ講座 研究室にて

張 偉

参考文献

[1] 正城 敏博 等, 「音声通信を考慮したマルチメディア ATM 通信方式と VLSI 化」, 信学技報, SSE97-10, 1997-04.

[2] 清水 敬司 等, 「ATM ブロック転送方式に基づくコネクションレス通信網構成法」, 信学技報, SSE97-156, 1997-12.

[3] 斉藤 忠夫, 「ポスト ATM 通信時代に向けてのネットワークアーキテクチャ」, 信学技報, SSE97-117, 1997-09.

[4] 松田 理絵子, 「マルチスレッド型 ATM 交換機アーキテクチャ」, 北陸先端科学技術大学院大学, 修士論文.

[5] Eiji OKI and Naoaki YAMANAKA, 「A High-speed ATM Switch Based on Scalable Distributed Arbitration」, IEICE Trans.Commun., VOL.E80-B, No-09, Sep.1997.

[6] パターソン&ヘネシー, 成田 光彰 訳, 「コンピュータの構成と設計」, 日経 BP 社.

[7] James R.Larus, 「SPIM S20: A MIPS R2000 Simulatior」, Computer Science Department, University of Wisconsin-Madison.

[8] Anne Rogers and Scott Rosenberg, 「Cycle Level SPIM」, Department of Computer Science, Princeton University.

[9] 青木 利晴, 青山 友紀, 濃沼 健夫, 「広帯域 ISDN と ATM 技術」, 社団法人 電子情報通信学会編

[10] David Wright 著, 岡山 博美 訳, 「マルチメディア通信と広帯域ネットワーク」, 日刊工業新聞社.

[11] 横河ディジタルコンピュータ株式会社 SI 事業部 著, 「ATM 入門」, トッパン（TOPPAN publishing）.

[12] 程時端 編著, 「総合業務数字網」, 中国通信学会主編・人民郵電出版社出版.

[13] Craig Partridge, Philip P. Carvey, 「A 50-Gb/s IP Router」, IEEE/ACM TRANSCTIONS ON NETWORKING, VOL.6 NO.3 JUNE 1998

[14] Galileo technology, 「GT-64120 System Controller For RC4650/4700/5000 and RM526X /527X/7000 CPUs」, www.galileoT.com

Appendix A：MIPS64 20Kc と GT-64120

MIPS64 20Kc プロセッサ

MIPS64 20Kc プロセッサは MIPS64 ファミリの中に非常に性能高いプロセッサの一つである。このプロセッサは６ステージのパイプライン処理であり、スタティック分岐予測と投機実行も行ない、ほとんどの命令が１クロックサイクルで実行され、1000MIPS の性能を持っている。

図 7.1で示すように、四つの実行ユニット、一つのメモリ管理ユニット、16KB 命令と 16KB ライトバックデータキャッシュ及び 32 個 64bit 汎用レジスタ（GPRs）と倍精度 64bit 浮動小数点レジスタが備えている。

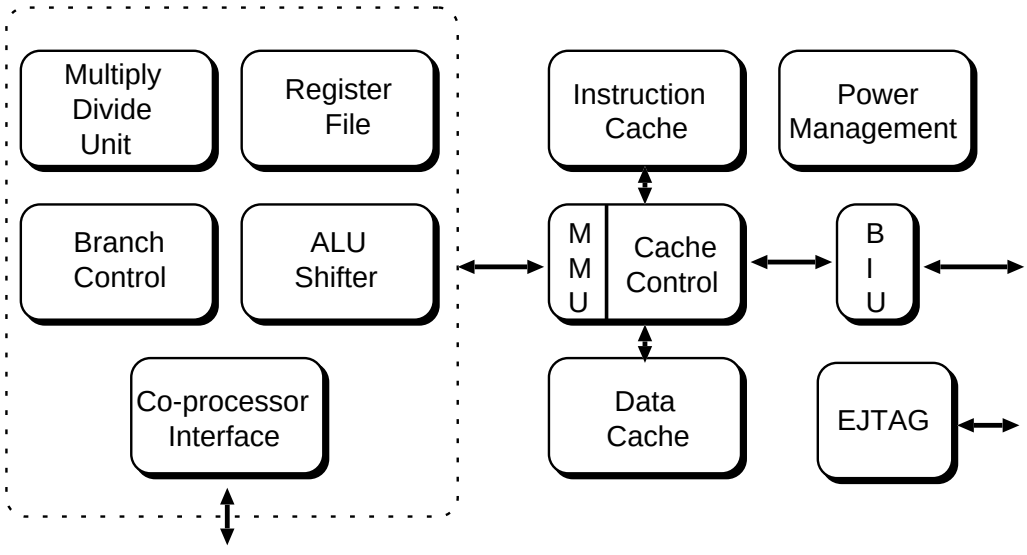


図 7.1: MIPS64 プロセッサ

GT-64120 システムコントローラ

GT-64120 は Galileo 会社に提供されたシステムコントローラである。GT-64120 は全ての高性能 64-bit バス MIPS CPU によるシステム構築に対して、シングル・チップ・ソリューションを提供している。

GT-64120 では、１チップで SDRAM コントローラ、デバイスコントローラ、４チャンネル DMA コントローラ、高速 PCI インタフェース、割り込みコントローラ等を備えている。

DMA 機能において、４つの独立のチャネル、リンクリストの設定によって連続実行機能を備え、特に Fly-By というデータ伝送機能をサポートしている。Fly-By とは DMA FIFO を通らずに二つのデバイスの間に直接データを伝送するという方式で、非常に高速データ伝送方式である。

GT-64120 には三つのバスアーキテクチャがある。

- CPU bus への 64-bit インタフェース（SysAD bus）
- メモリとデバイスサブシステムへの 64-bit インタフェース（LocalAD bus）
- 二つ独立な 32-bitPCI インタフェースもしくは一つ 64-bitPCI インタフェース

三つのバスが独立なので、同時動作ができる。SysAD bus の最大動作周波数は 75MHz であるので、CPU は 600MByte/sec でのデータ転送ができる。また、全てのリソースへの CPU アドレスのリマッピングをサポートしている。SDRAM コントローラは 64-bit 幅の LocalAD bus を通じて、いくつか型の SDRAM をサポートしている。最大アドレスイングが 512MByte に達し、最大動作周波数は 75MHz できる。UMA （unified memory achitecture）特性により、外部マスタは SDRAM に直接アクセスもできる。PCI インタフェースの最大動作周波数は 66MHz であり、これを介して周辺デバイスと繋ぐ。

DMA 機能は PCI、メモリ、デバイスの間にデータを伝送することである。ソースとデステネイションはバイトアドレス境界を指定できる。二つの 64-Byte 内部 FIFO により、二つの伝送を同時に行なうこともできる。ローカルデータバスに繋がる装置に Fly-By 機能をサポートしている。

Appendix B：シミュレータの動作検証

仲裁ロジックの検証

４クラス全部同じフレーム長とし、固定時間間隔で発生して、仲裁ロジックの動作を確認する。

出力回線速度は155Mbps（155Mbps=19.375Byte/μsec）で、１個57Byteフレームの出力時間は約3μsecである。

４クラス全部12μsec間隔で発生する場合、全てのキュー長が０である。これは出力速度が回線速度と同じなので、つまり回線利用率はちょうど100%であるからである。

４クラス全部10μsec間隔で発生する場合、つまり出力回線速度より20%越える。この場合、バッファ１、２、３の到着フレームが合わせて全ての回線利用率を占めているというのとである。この時のキューの成長を図7.2と図7.3で示す。図7.2では、制限時間0であるから、全て優先度による出力なので、バッファ１、２、３フレームが溜っていない。バッファ０だけフレームが溜っていく。つまりバッファ０が一つの組、バッファ１、２、３がもう一つの組になってしまうのである。一方、制限時間500μsecである図7.3を見ると、最初バッファ１、２、３は優先度が高いので、ほとんど溜っていない。バッファ０は出力機会が得られないので、キュー長が伸びている。ある時間が経って、バッファ０内のフレームの待ち時間が制限時間を越えることになる。すると、バッファ０優先度が高くなった。この時バッファ１の優先度が一番低くなった。続いてバッファ１のキュー長が伸びる。ある時間が経って、バッファ１内のフレームの待ち時間も制限時間を越えることになった。こういうふうに段々進行すると、最後全てのバッファが制限時間を越えることになる。それで最初の状態と同じように、バッファ１、２、３のキュー長が安定し、バッファ０のキュー長が増加の一方になった。

図7.4と図7.5は４クラス全部6μsec間隔で発生する場合、つまり出力回線速度より100%越える場合である。この場合、バッファ２、３の到着フレームが合わせて100%の回線利用率を占めている。これらの図は先の二つの図と理由が同じであるから、説明を省く。だた、今回はバッファ０、１とバッファ２、３二つの組になった、この原因はバッファ２、３の到着フレームが全ての回線利用率を占めているからである。

この結果から見れば、仲裁ロジックは想定したとおりの動作を果たしていると考えられる。注意が必要の点は、ここでの実験は仲裁ロジックの動作を確かめるために、極端な条件で行なったことである。実際に回線利用率が100%を越える場合はない、その上、フレームの到着はランダムであり、固定間隔ではないので、このようなキュー長が線形的無限に大きくなる可能性はない。

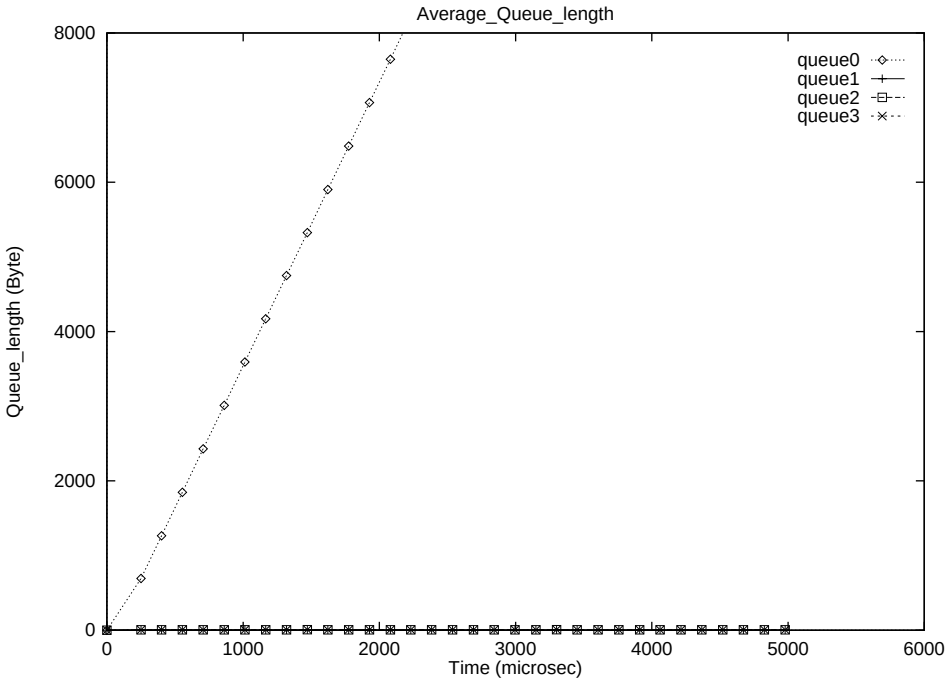


図 7.2: 発生間隔 10μsec、制限時間 0 の平均キュー長

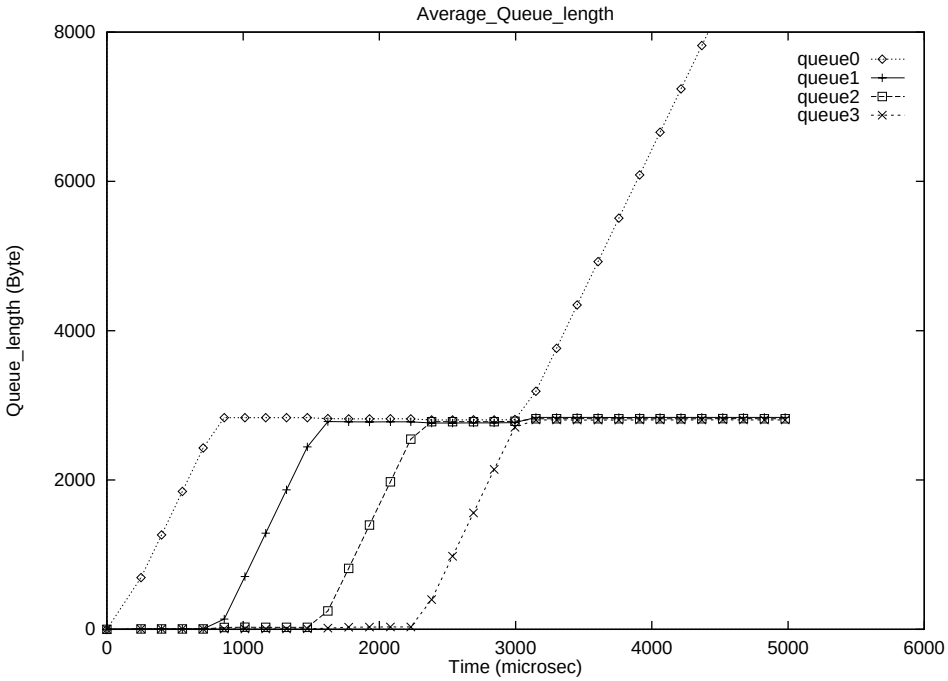


図 7.3: 発生間隔 10μsec、制限時間 500μsec の平均キュー長

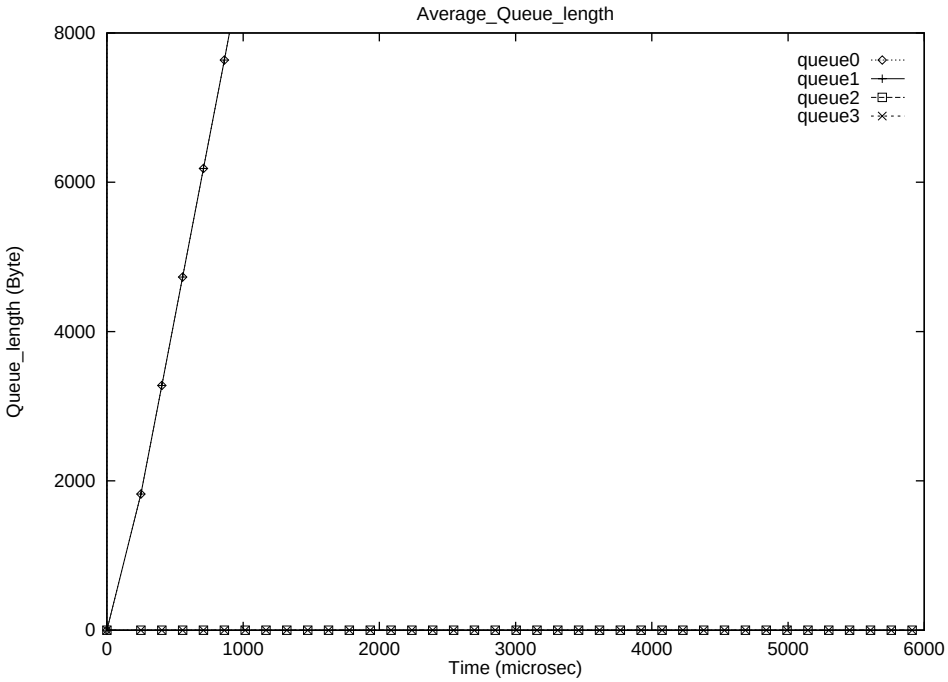


図 7.4: 発生間隔 6μsec、制限時間 0 の平均キュー長

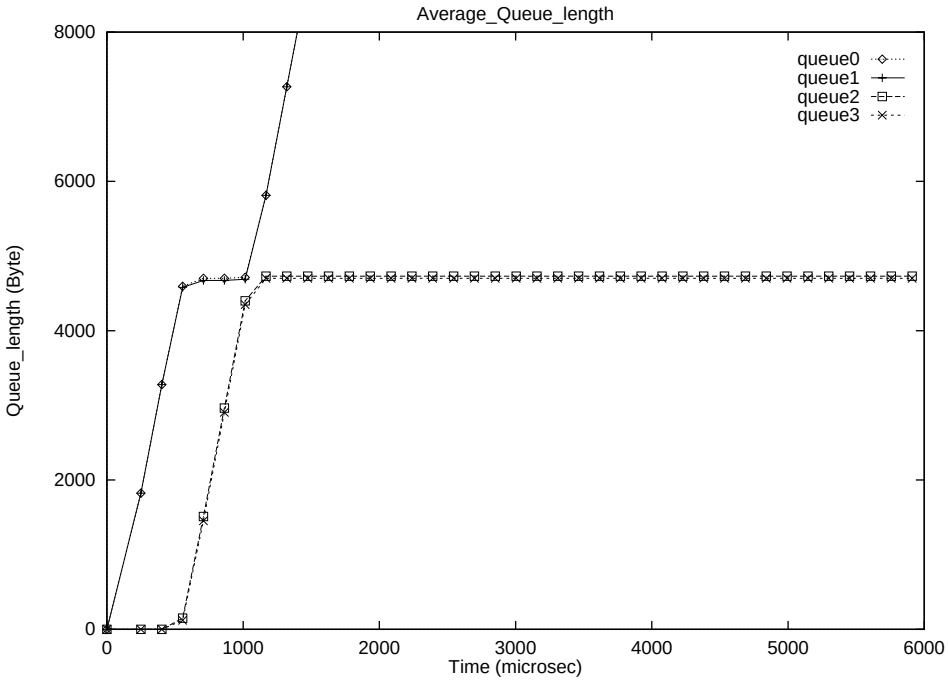


図 7.5: 発生間隔 6μsec、制限時間 500μsec の平均キュー長

平均キュー長の測定値と理論値の比較

発生源においてフレーム長 1Byte として、平均発生間隔時間は 1000μsec と固定して、出力回線速度は（その一個フレームを出力する所要時間）100μsec から 900μsec まで変化させた時のレポートされたキュー長について観測を行なう。その数字が選ばれたのは計算することが簡単のためである。フレームの発生間隔を指数分布に従って、発生個数は 1000000 個、サンプリング間隔時間は 100μsec として、サンプリング時間は 10000μsec である。得られる結果が M/D/１の待ち列モデルの理論値と比べる。

M/D/１の待ち列における平均キュー長 L_qは、利用率をρとして、

$$L_q = \frac{\rho}{2 \times (1-\rho)} \cdot \lambda \cdot \overline{t_s} = \frac{\rho}{2 \times (1-\rho)} \cdot \frac{\lambda}{\mu} = \frac{\rho^2}{2 \times (1-\rho)}$$

で与えられる。式の中に$\overline{t_s}$は平均サービス時間、μは平均サービス率、λは平均到着率である。

結果を表 7.1で表す。曲線を図 7.6で示す。

利用率 ρ	理論値 L_q	測定値 L_q	差異 ＋ -	測定値 瞬間最大キュー長（個）
0.1	0.005556	0.007127	0.001571	4
0.2	0.025000	0,030928	0.005928	5
0.3	0.064285	0.077973	0.013688	5
0.4	0.133333	0.154981	0.021647	7
0.5	0.250000	0.277828	0.027828	9
0.6	0.450000	0.477529	0.027529	10
0.7	0.816667	0.836692	0.020025	17
0.8	1.600000	1.598258	-0.00174	23
0.9	4.050000	4.080100	0.030100	41

表 7.1: 平均キュー長の実測値と理論値の比較

この表と図から見ると、理論値と測定値がほぼ一致していることがわかった。誤差が起こった原因とは、乱数発生の誤差等が考えられる。また本シミュレーションの基本時間単位はμsec となるので、シミュレーションの間に計算した小数は全部捨てられ、整数だけ保留したので、誤差も生じる。しかし、これらの誤差はシミュレーションの結果に対して、大きな影響がないと考えられる。

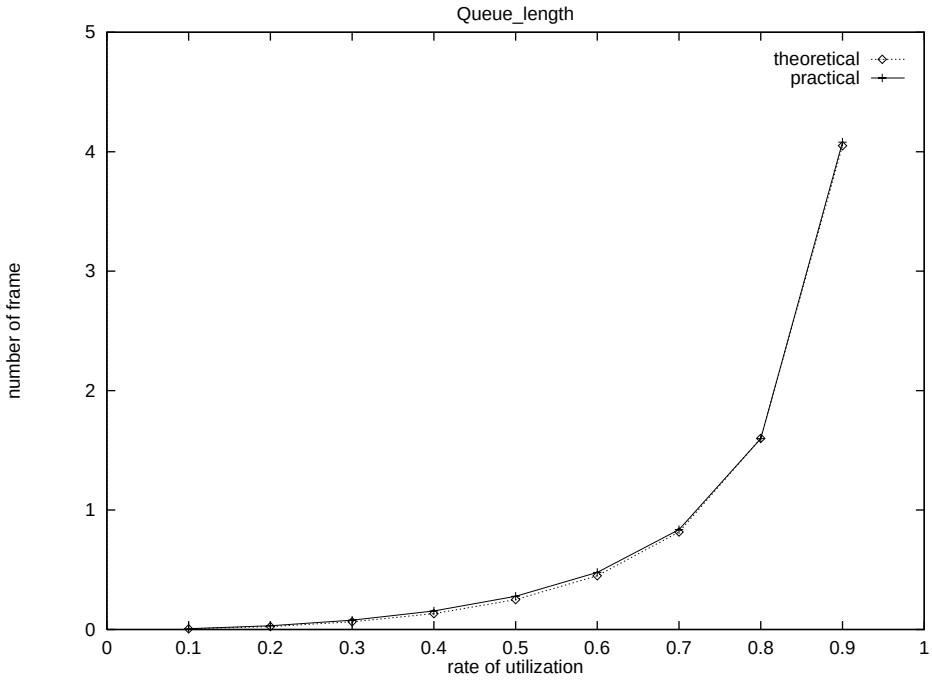


図 7.6: 平均キュー長の実測値と理論値の比較

Appendix C：通常運用状況のシミュレーションの補足

各クラスの回線利用率は偏りの場合

1. クラス 1、2 は 70%を占める、クラス 3、4 は 10%を占める。

   - 入力条件
     入力パターンを表 7.2で示す。

クラス	フレーム長（Byte）	平均発生間隔（μsec）	発生間隔分布	相応する利用率
クラス１	4500	4645	指数分布	0.05
クラス２	1518	1567	指数分布	0.05
クラス３	192	30	固定間隔	0.33
クラス４	57	8	固定間隔	0.367

表 7.2: パターン B-1

   - 結果
     制限時間は 0 と 500μsec の場合の平均キュー長の変化を図 7.7、図 7.8で示す、結果データを表 7.3と表 7.4でまとめる。

2. クラス 1、2 は 10%を占める、クラス 3、4 は 70%を占める。

   - 入力条件
     入力パターンを表 7.5で示す。
   - 結果
     制限時間は 0 と 500μsec の場合の平均キュー長の変化を図 7.9、図 7.10で示す、結果データを表 7.6と表 7.7でまとめる。

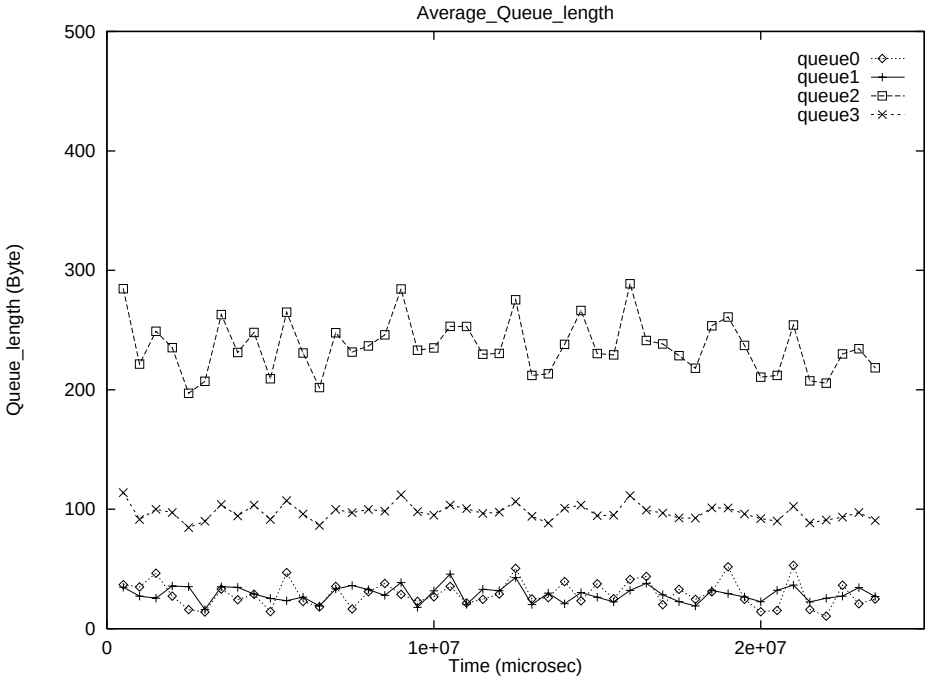


図 7.7: 制限時間 0、パターン B-1 の平均キュー長

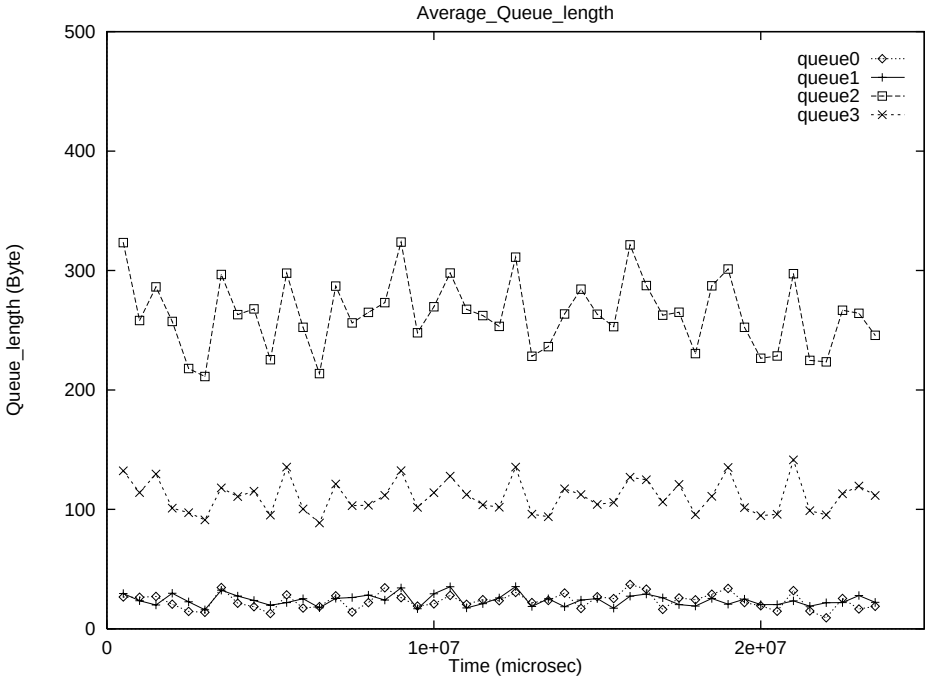


図 7.8: 制限時間 500μsec、パターン B-1 の平均キュー長

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	5152	28.96	13500(3)	153	3281
クラス２	15235	29.05	7590(5)	143	2491
クラス３	800000	236.78	5184(27)	63	864
クラス４	3000000	97.40	3249(57)	24	468

表 7.3: 制限時間 0、パターン B-1 の結果データ

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	5152	23.21	13500(3)	126	1945
クラス２	15235	24.02	7590(5)	124	1285
クラス３	800000	263.84	7296(38)	67	1190
クラス４	3000000	111.10	5187(91)	26	743

表 7.4: 制限時間 500μsec、パターン B-1 の結果データ

クラス	フレーム長（Byte）	平均発生間隔（μsec）	発生間隔分布	相応する利用率
クラス１	4500	663	指数分布	0.35
クラス２	1518	224	指数分布	0.35
クラス３	192	198	固定間隔	0.05
クラス４	57	59	固定間隔	0.05

表 7.5: パターン B-2

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	266023	1219.02	63000(14)	502	8040
クラス２	792097	719.73	18216(12)	306	1506
クラス３	893940	62.64	576(3)	195	980
クラス４	3000000	70.09	570(10)	120	698

表 7.6: 制限時間 0、パターン B-2 のの結果データ

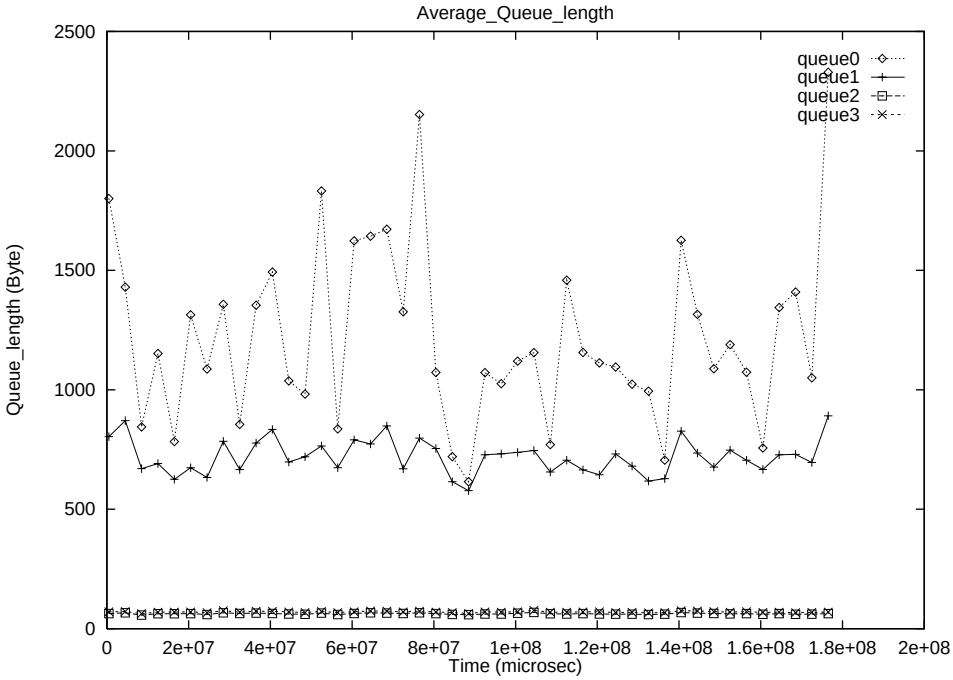


図 7.9: 制限時間 0、パターン B-2 の平均キュー長

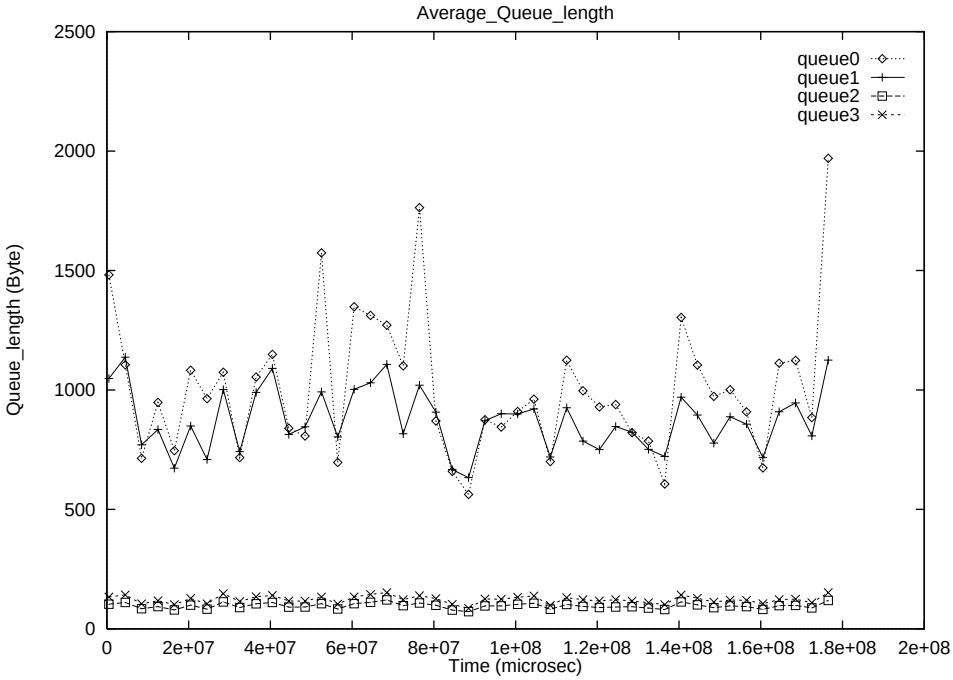


図 7.10: 制限時間 500μsec、パターン B-2 の平均キュー長

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	266023	1008.58	58500(13)	450	7502
クラス２	792097	873.16	19734(13)	338	1774
クラス３	893940	96.19	960(5)	248	1233
クラス４	3000000	122.64	855(15)	181	966

表 7.7: 制限時間 500μsec、パターン B-2 のの結果データ

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	37427	629.00	40715	460	7998
クラス２	111291	418.30	14005	314	2495
クラス３	879981	394.93	6068	158	1270
クラス４	3000000	228.12	3338	71	751

表 7.8: 標準偏差 0.1、制限時間 0、両方ともランダム場合の結果データ

フレーム長と発生間隔と共にランダム発生する場合

- 入力条件
  フレーム発生間隔は４クラス全部指数分布に従う。フレーム長は４クラス全部対数正規分布に従って、標準偏差は 0.1 と 0.5 二つの場合のシミュレーションを行なった。平均フレーム長と平均発生間隔はパターン１と同じである。従って、４クラスの平均相応利用率はそれぞれ 20%である。

- 結果
  標準偏差は 0.1 の平均キュー長の変化を図 7.11、図 7.12で示す、結果データを表 7.8と表 7.9でまとめる。標準偏差は 0.5 の平均キュー長の変化を図 7.13、図 7.14で示す、結果データを表 7.10と表 7.11でまとめる。

フレーム長だけランダム発生する場合

- 入力条件
  フレーム長は４クラス全部対数正規分布に従って、標準偏差は 0.1、0.5、1.5 三つの場合のシミュレーションを行なった。平均フレーム長がパターン１と同じであり、４クラスの平均

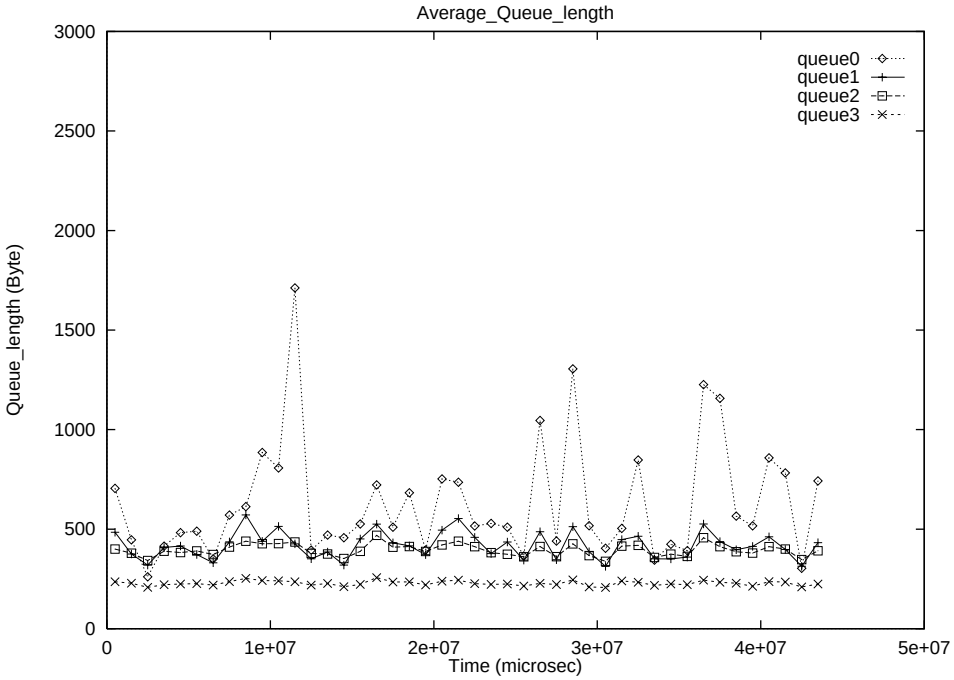


図 7.11: 標準偏差 0.1、制限時間 0、両方ともランダム場合の平均キュー長

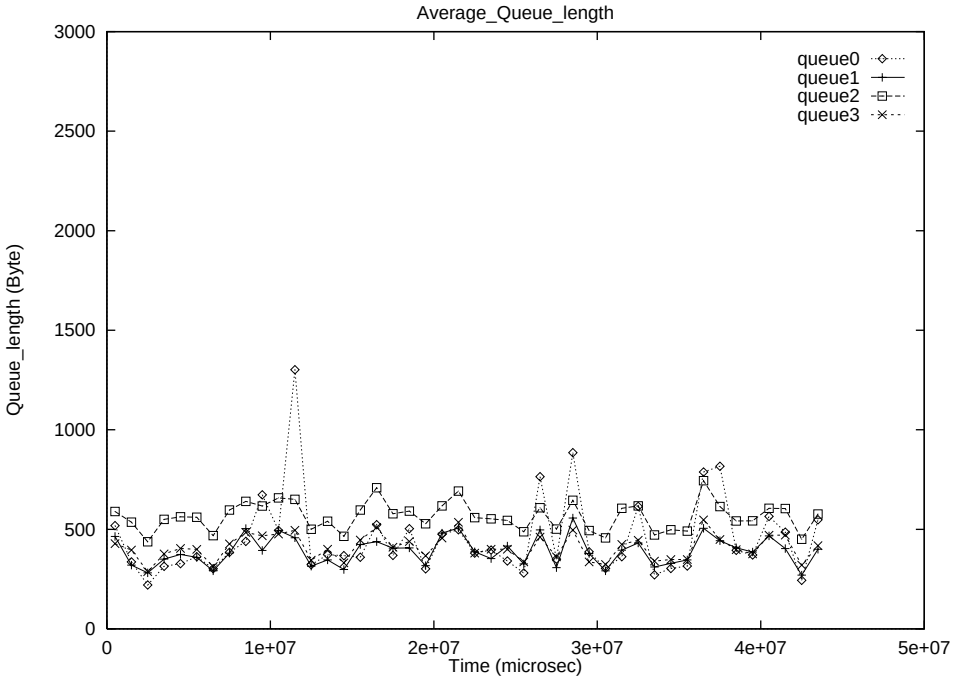


図 7.12: 標準偏差 0.1、制限時間 500μsec、両方ともランダム場合の平均キュー長

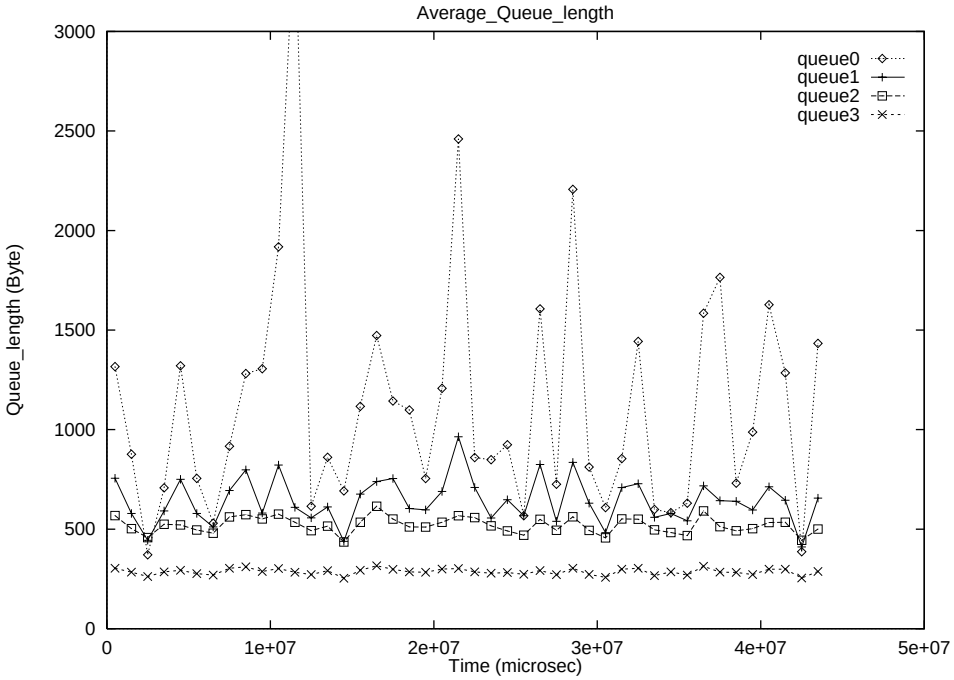


図 7.13: 標準偏差 0.5、制限時間 0、両方ともランダム場合の平均キュー長

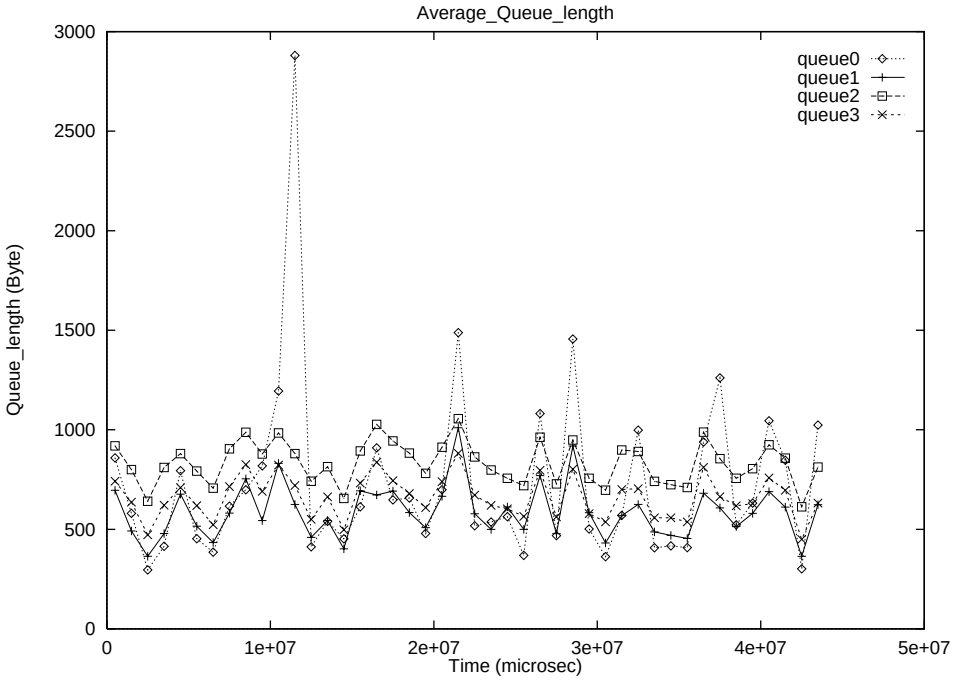


図 7.14: 標準偏差 0.5、制限時間 500μsec、両方ともランダム場合の平均キュー長

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	37427	453.12	36875	367	7502
クラス２	111291	391.29	16972	300	1774
クラス３	879981	565.92	7193	205	1233
クラス４	3000000	410.14	5714	116	966

表 7.9: 標準偏差 0.1、制限時間 500μsec、両方ともランダム場合の結果データ

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	37427	1121.39	60795	714	11539
クラス２	111291	642.60	20586	422	2701
クラス３	879981	519.63	7164	188	1416
クラス４	3000000	286.17	3949	77	790

表 7.10: 標準偏差 0.5、制限時間 0、両方ともランダム場合の結果データ

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	37427	729.92	52314	525	10277
クラス２	111291	588.08	24039	396	3085
クラス３	879981	834.12	9386	270	1573
クラス４	3000000	661.01	6401	158	1033

表 7.11: 標準偏差 0.5、制限時間 500μsec、両方ともランダム場合の結果データ

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	39832	41.80	5120	49	189
クラス２	117836	130.45	1984	112	493
クラス３	925540	197.50	1558	94	461
クラス４	3000000	157.43	1298	57	357

表 7.12: 標準偏差 0.1、フレーム長ランダム場合の結果データ

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	48148	49.39	5120	56	487
クラス２	127190	133.65	5120	111	505
クラス３	998091	206.50	2161	99	649
クラス４	3000000	163.92	1674	61	517

表 7.13: 標準偏差 0.5、フレーム長ランダム場合の結果データ

相応利用率はそれぞれ 20%である。しかし、発生間隔はフレーム長と回線利用率により決めるものである。つまり、１個フレームを発生した後、そのフレームの長さと当該クラスの回線利用率によって、次のフレームの発生の間隔を決める。

- 結果
制限時間 0 の結果だけを示す。標準偏差は 0.1、0.5、1.5 の場合、キュー長の変化を図 7.15、図 7.16と図 7.17で示す、結果データを表 7.12、表 7.13 と表 7.14でまとめる。

クラス	発生個数	平均キュー長（Byte）	瞬間最大キュー長（Byte）	平均待ち時間（μsec）	最大待ち時間（μsec）
クラス１	91333	105.14	6240	121	1011
クラス２	163475	146.64	6201	144	1079
クラス３	1021063	282.90	6604	138	929
クラス４	3000000	249.33	5711	90	726

表 7.14: 標準偏差 1.5、フレーム長ランダム場合の結果データ

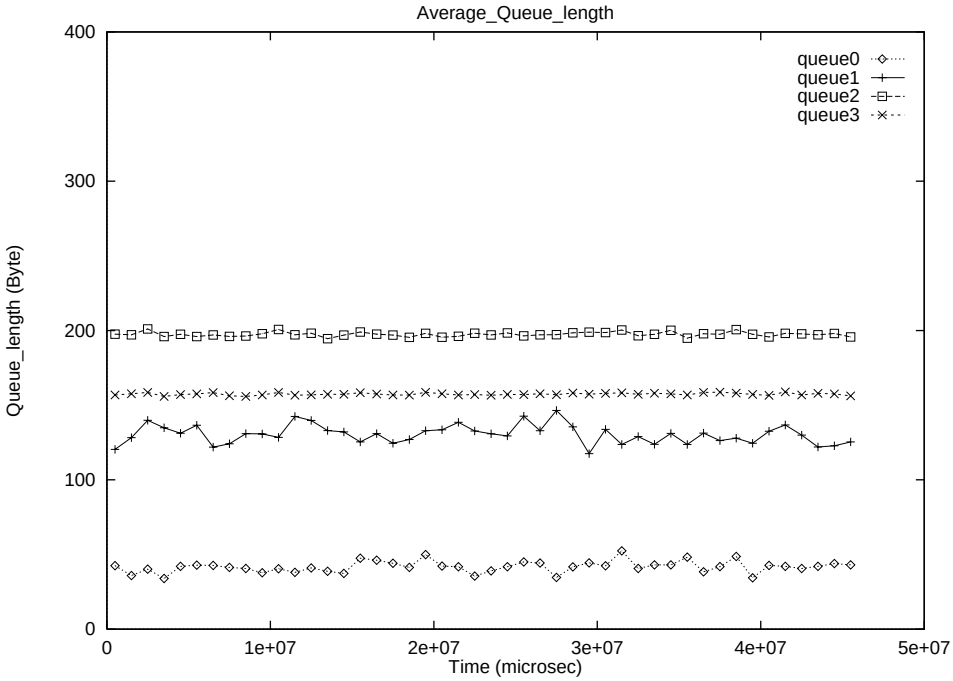


図 7.15: 標準偏差 0.1、フレーム長ランダム場合の平均キュー長

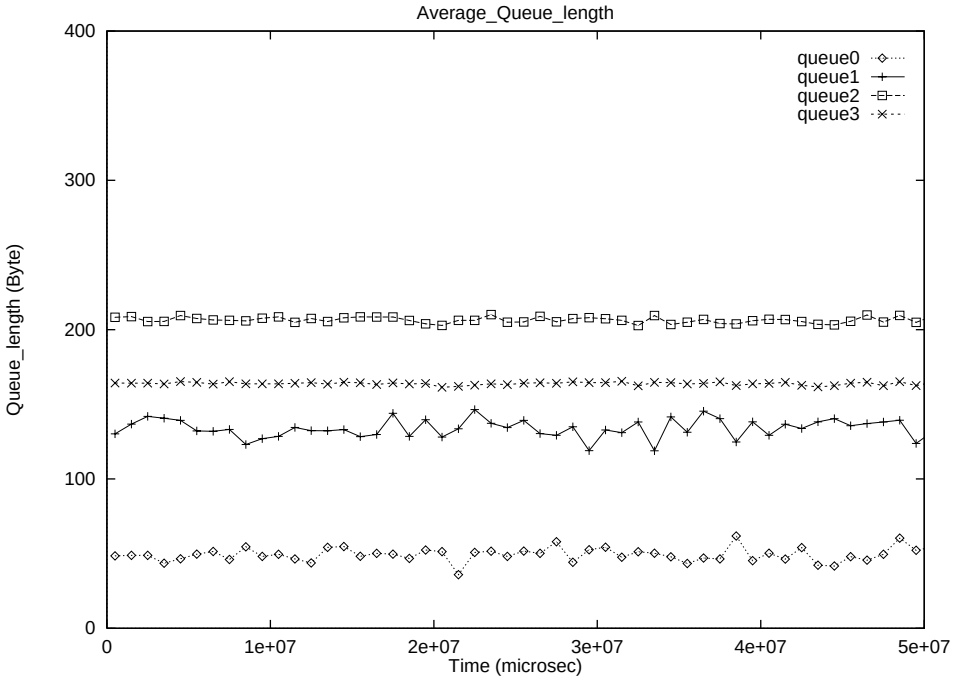


図 7.16: 標準偏差 0.5、フレーム長ランダム場合の平均キュー長

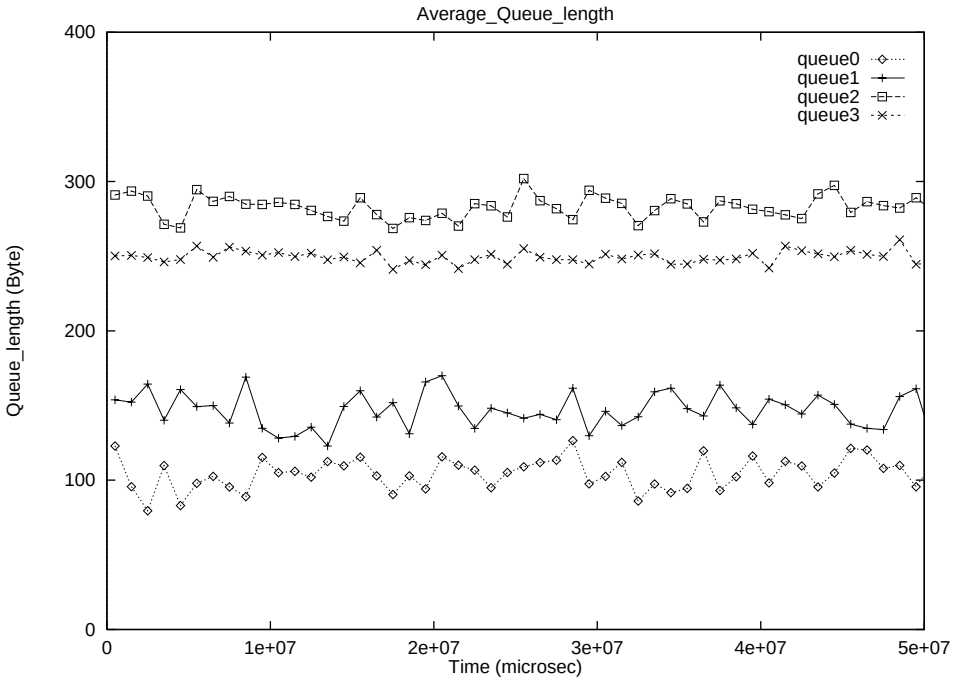


図 7.17: 標準偏差 1.5、フレーム長ランダム場合の平均キュー長